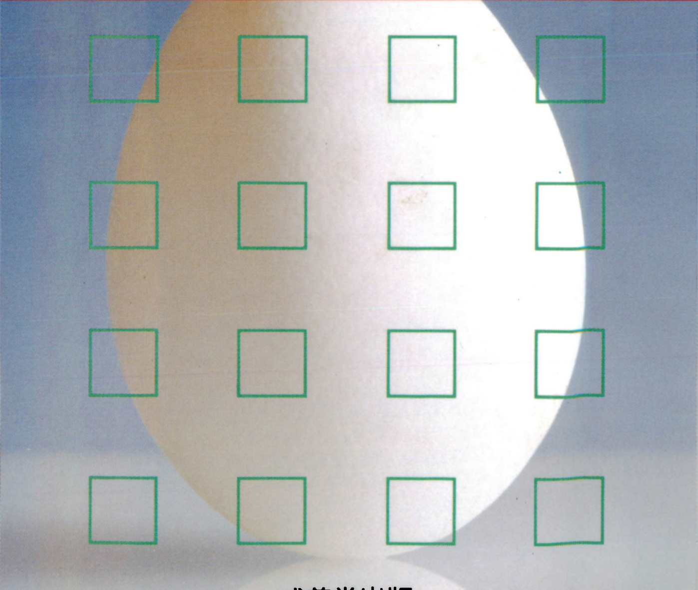
PC-98シリーズ

BASIC 用語・用例辞典

伊東ひろみ著



成美堂出版

PC-98シリーズ
BASIC用語・用例辞典
成美堂出版

PC-98シリーズ

BASIC
用語・用例辞典

伊東ひろみ著

成美堂出版

PC-98シリーズ BASIC 用語・用例辞典

伊東ひろみ著

成美堂出版

はじめに

さて、どんなことに使おうかといろいろと夢をもって購入したパソコン。いざ使いはじめてみると、ゲームで遊んだり、市販ソフトやワープロとして利用するだけではおもしろくない。もっと自分で活用したいと思っている人も多いことでしょう。

アプリケーションソフトを利用するだけではものたりなく感じて、自分専用のソフトがほしいと思っている人もたくさんいるはずです。しかし、個人では専門のプログラマーに自分専用のプログラムを開発してもらうわけにもいかないでしょう。

本書は、このようなプログラムを自作してみたいというたくさんの方々に、DO IT YOURSELFでプログラムを作る楽しさを知っていただくために、単なるコマンドリファレンスとしてだけではなく、プログラムの勉強にもなるように少しくふうされています。

用語解説には、基本的な使用法だけでなく、応用例としてのサンプルプログラムを用意しています。これらは入力も簡単で、小さな、テストランのしやすいプログラムの形にして、なるべくアミューズメント性のある楽しいものや、実用性のあるものを採用しています。

これらの小さなサンプルプログラムをいくつも作っていくことによって、BASICの学習もできます。

また、コマンド・ステートメントが機能別に配列されているので、本の初めから終わりまで通読していっても勉強になるでしょう。

本書は、BASICの入門者・初心者の方がパソコンの横にいつも置いてすぐに参照できるよう、またポケットの中に入れていつでもどこでもBASICの勉強ができるように考えて作られています。

本書は、

●BASICの入門者・初心者

●BASICの基本命令を実際にパソコンで試しながら覚えたい人

●BASICのプログラムを作りながら、わかりやすい辞典として使いたい人

●BASICプログラムを作りたいが、よいアイデアがうかばない人

●マニュアルを読みこなすのがめんどうな人

●マニュアルでは難しすぎてわからない人

このような方々に、

◎初めから読み通してBASICの勉強をする

◎おもしろそうなサンプルプログラムを打ち込んでみる

◎役に立ちそうなサンプルプログラムを打ち込んでみる

◎マニュアルの代わりに通読する

◎リファレンスマニュアルの代わりに手元に置いてプログラミング中にひく

◎プログラムの作り方やコマンド・ステートメントの使用方法を知る

◎自分の作りたいプログラムのアイデアの種を探す

このように利用していただければ幸いです。

伊東ひろみ

本書の使い方

本書は、たくさんあるBASICのコマンドやステートメント(文)、関数の中から、比較的よく使われるものを選んで、コンパクトな辞典として編集されています。

●入門書として使う

各命令は、種類ごとに分類して配列され、入門者から初心者を対象に解説が加えられています。本書の初めから終わりまでを通して読み、用例のプログラムを打ち込んで動きを確認したり、試したりしながら、BASICの基本的な命令を勉強することができます。

●辞典として使う

本書巻末のアルファベット順の索引を利用して、使い方のわからない命令やパラメータの指定などをすばやく調べることができるリファレンスマニュアルとして使うことができます。

●プログラムの自作に役立てる

命令の一部には、初歩的な応用プログラムがついています。これらのプログラムをもとに、より役立つ、楽しいプログラムに改良して、BASICプログラミングの腕前をあげてみてはいかがでしょう。

●プログラム入力の注意

紙面の関係上、プログラムリストの1行が2行に分かれている部分があります。この部分は、続けて1つの行として打ち込んでください。また、そのとき2行目のプログラムの先頭にスペースがあるときは、そのままあけてスペースとして打ち込んでください。

```
440 ON SL GOSUB *INPT,*CRCT,*DELT,*DSPL,*DLST,*
SRCH,*QUIT
```

↑ 続けて入力する

```
120 FIELD #1,20 AS MOF$,2 AS SEF$,4 AS TAF$,8 AS
 BAF$
```

↑ 本来の1行のプログラムにある1文字分のスペースである。このスペースを含めて続けて入力する

もくじ

BASICの文法と編集

1.BASICの文法

2.BASICプログラムの編集

PC-98シリーズ BASIC用語・用例辞典

1.基本コマンド

2．基本入出力命令

3. 実行制御

4.変数定義

5.テキスト画面制御

6.グラフィック&サウンド制御

7.ファイル制御

8.例外処理命令

9.文字列操作

10.数値関数

BASICの文法と編集



1. BASICの文法

❶ PC-98シリーズのBASIC

●プログラム自作のためのN_{88}－日本語 BASIC (86)

PC－98 シリーズは、パソコンの中でももっともポピュラーなパソコンとして、多くの会社や家庭で使われています。さまざまなしごとに、ホビーにと、おおいに活用されていますが、最近では、目的に合ったアプリケーションソフトを購入して使用するのが一般的な使い方になっています。

しかし、こうした既製のソフトではなく、自分専用の小回りのきく小さなプログラムを自作して、個人的な情報を整理したり、ちょっとしたしごとをさせるようにすれば、パソコンの楽しみ方や使い方にもずっと広がりがでてきます。

このようなプログラムを作るため、旧タイプの PC－98 シリーズには、本体とともに「N_{88}－日本語 BASIC (86)」という BASIC が付属しています。この「N_{88}－日本語 BASIC (86)」には、ROM－BASIC と DISK－BASIC という、2つの BASIC が含まれています。ROM－BASIC は、パソコン本体内に取り付けられているものです。最近の PC－98 シリーズでは、DISK－BASIC は別売となっています。本書では、このうち機能の高い DISK－BASIC を基本にとりあげています。

なお、PC－98 シリーズ用の BASIC には、「N_{88}－日本語 BASIC (86)」のほかに、「N_{88}－日本語 BASIC (LT)」や、市販の MS－DOS 上で動作する「MS－DOS 版 N_{88}－日本語 BASIC (86) インタプリタ、コンパイラ」、その他各種の市販 BASIC があります。それぞれ BASIC に違いがあるので、混乱しないように注意してください。

❷ BASICの文

●プログラムのもっとも小さい構成単位

BASICは、数ある「プログラム言語」の中のひとつです。

もともとコンピュータは、電流がプラスとマイナスのどちらにあるかを使って、種々の計算やしごとをしています。このコンピュータにしごとをさせるには、人間が命令を与えなければなりません。しかし、電流の＋、－を直接指示していたのではとてもふつうの人がコンピュータを使いこなすわけにはいきません。

そこで、プログラム言語という人間にもわかりやすい、コンピュータにしごとをさせるためのことばを作りだして、このことばを使ってコンピュータに命令を与えているわけです。この命令の集まりをプログラムといいます。

プログラム言語は、機械を相手にしたことばですので、文法が細かく決められています。

まず、ことばの単位は単語ですが、BASICではこれを「命令」と呼びます。

命令には、「コマンド」、「ステートメント」、「関数」などの種類がありますが、これらについては本書の用語・用例辞典の中でひとつひとつ説明していますので、ここでは、「文」からとりあげましょう。
「文」は、プログラムのもっとも小さい構成単位です。
「文」は、1つの「命令」と、この命令に与える情報である「オペランド」から成り立っています。「オペランド」には、定数、変数、関数などを指定することができます。

▶「文」＝「命令」＋「オペランド」 例　PRINT 1234

例は、「PRINT」という命令と「1234」というオペランドによって、ひとつの文が構成されていることを示しています。

❸ダイレクトモード

●キーボードから入力した文をその場で実行する

BASICには、2つのモードがあります。「ダイレクトモード」と「プログラムモード」です。

「ダイレクトモード」は、キーボードから打ち込んだ文をただちに実行するモードです。例のようにキーボードから文を打ち込むと、命令をただちに実行します。

例	PRINT	1234+1234 ⏎

「PRINT」は、オペランドの結果を画面に表示する命令です。

オペランドには「1234+1234」の計算が指定されています。

「⏎」は、キーボードのキャリッジリターンキーを押すことを示しています。

パソコンは、この⏎キーが押されるたびに、1つの文などの実行単位を示されたと解釈しています。このため、パソコンに何かを実行させるには、必ず⏎キーを押さなければなりません。今後、特に必要なところ以外は、⏎キーのマークは付けませんが、一連の命令や文を入力した後は、必ず⏎キーを押してください。

さて、ダイレクトモードで上の命令文を入力し、⏎キーを押すと、

結果	2468

とディスプレイに表示されます。

これは、命令によってオペランドの計算を行ない、ディスプレイに表示されたことを示しています。いくつか試してみてください。

❹プログラムモード

●行番号が付けられた複数の文をパソコンが順番に実行

「ダイレクトモード」では、電卓的にひとつひとつの命令を実行していくだけですが、これではパソコンをコンピュータとして使っているとはいえません。

「プログラムモード」は、複数の文を並べて、パソコンが順番に命令を実行していくモードです。

BASICでは、文に行番号を付けて順番に並べていくことによって、1行ごとに命令の実行順序を示しています。一度入力されたプログラムは、消去してしまうまでパソコンのメモリに記憶されていますので、何回でも同じプログラムを実行することができます。

また、メモリ上に記憶されたプログラムは、フロッピーディスクなどの外部記憶装置に記録しておくことができます。外部記憶装置に記録されたプログラムは、いつでもパソコンのメモリ上に読み込んで実行することができるので、何回も同じプログラムを作る必要がなくなります。一度作成したプログラムを改良して、さらに充実したプログラムに拡張することもできます。

このようにして、自分で作ったBASICプログラムを活用していくのです。



❺ BASICの行

●プログラムの実行順序を示す

BASICのプログラムは、「行」の集まりで構成されています。「行」は、一般的に1つの文からできています。「行」の先頭には「行番号」を付けて、プログラムの実行順序を指定します。

> ▶「行」＝「行番号」＋「文」
>
> 例　100　PRINT　1234 ⏎

この例では、「『100』という行番号が付けられ、『1234』という文字列をディスプレイに表示する」というプログラムが、メモリ上に登録されます。

一度登録されると消去されるまでメモリ上に残っていますので、「LIST」コマンドで内容をディスプレイに表示して、確認することができます。

また、「マルチステートメント」といって、1行のプログラム中に複数の文を書くこともできます。この場合は、文と文とを「：（コロン）」で区切り、最大254文字までの範囲で書きます。1行の中では、左の命令から順番に実行されていきます。

> 例　110　LOCATE　10, 10 : PRINT　1234

110行に、ディスプレイの「10，10」の位置から「1234」を表示するプログラムを登録しました。

❻ 行番号

●編集や修正をするときのために10おきに付ける

BASICのプログラムで付けることのできる行番号は、1～65529です。「－（マイナス）」や「．（小数点）」の入らない整数で指定しなければなりません。

プログラムは行番号の小さい順に実行されていきます。「GOTO」文や「IF ～ THEN ～ ELSE」文などでプログラムの実行をとばしたり、分岐したりするのも行番号を使って行ないます。

行番号は、プログラムをメモリ上に登録したり、フロッピーディスクに記録する順番を示すための番号でもあります。

また、行番号はプログラムを作成したり、手直しをする編集のときにも使います。プログラムの修正やメモリ上への登録は1行ごとに行なわれます。

プログラムを作成したり、編集、修正をする際に、行と行の間にいくつかの命令を割り込ませたいことがよくあります。このように行の挿入をする必要が出た場合に、行番号を1、2、3のように詰めて並べてしまうと、挿入ができなくなってしまいます。

一般的に、プログラムの行番号は、10、20、30というように、10ずつ増やして付けられているのがふつうです。こうすれば次の行番号との間に、簡単に9行までのプログラムを挿入することができます。ある程度プログラムができあがってきたり、挿入する余裕がなくなった場合は、「RENUM」コマンドを使って、もう一度10行おきに行番号を付けなおします。

❼キャラクタ

●パソコン内部処理用のアスキーコードが付いて

BASICで使用することのできるキャラクタは、アルファベット大文字、アルファベット小文字、数字、特殊記号、グラフィックキャラクタ、日本語文字です。

アルファベットの大文字と小文字は、「 ”(ダブルクォーテーション)」に囲まれた「文字列」としては区別されますが、プログラムそのものの中では区別されません。

キャラクタには、パソコン内部で処理されるための番号が付いて

います。これをアスキーコードといいます。16進数で表わされたキャラクタのアスキーコード表を次に示します。

■アスキーコード表

上位4ビット→ / 下位4ビット↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D_E		0	@	P		p				ー	タ	ミ		╳
1	S_H	D_1	!	1	A	Q	a	q			。	ア	チ	ム		円
2	S_X	D_2	"	2	B	R	b	r			「	イ	ツ	メ		年
3	E_X	D_3	#	3	C	S	c	s			」	ウ	テ	モ		月
4	E_T	D_4	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	E_Q	N_K	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	A_K	S_N	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	B_L	E_B	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	B_S	C_N	(	8	H	X	h	x			ィ	ク	ネ	リ	♠	
9	H_T	E_M	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	♥	
A	L_F	S_B	*	:	J	Z	j	z			ェ	コ	ハ	レ	♦	
B	H_M	E_C	+	;	K	[	k	{			ォ	サ	ヒ	ロ	♣	
C	C_L	→	,	<	L	¥	l	¦			ャ	シ	フ	ワ	●	
D	C_R	←	−	=	M	]	m	}			ュ	ス	ヘ	ン	○	
E	S_O	↑	.	>	N	∧	n	~			ョ	セ	ホ	゛		
F	S_I	↓	/	?	O	_	o				ッ	ソ	マ	゜		

▶アスキーコード表の見方

「A」のアスキーコードは「上位4ビット＝4、下位4ビット＝1」、両方を合わせて「41」が16進数のコードです。この「A」の文字をCHR$関数を使って画面に表示するには、

```
PRINT  CHR$(&h41) ⏎
```

と入力します。

❽特殊記号

●特別な機能を持ったBASICプログラム用の記号

キャラクタのうち、BASICで特別な機能を持ったものを特殊記号といいます。以下に、記号とその意味を示します。

記号	読み方	意味
'	アポストロフィー	REM文（注釈文）の代わりの記号。
?	クエスチョンマーク	PRINT文の代わりの記号。
;	セミコロン	PRINT文の中などで、データの区切りとして用いる。
,	カンマ	データの区切りや、パラメータの区切りとして用いる。
:	コロン	マルチステートメントの命令の区切りとして用いる。
−	マイナス	LIST、DELETEコマンドなどで、行の範囲を示すのに用いる。
.	ピリオド（小数点）	LIST、EDITなどで、BASICインタプリタの管理する現在の実行位置（エラーが発生したり、訂正した行＝ポインタ）を示す。
*	アスタリスク	ラベルの先頭に付けて、ラベルであることを示す。
	スペース（空白）	「"（ダブルクォーテーション）」で囲まれた文字列など以外では、スペースは無視して実行されるが、命令の直後には必ずスペースをおかなければならない。

❾定　数

●それぞれに付属すべき記号に注意

定数には次のものがあります。

文字定数

255文字以内の「”（ダブルクォーテーション）」で囲まれたアルファベット、カナ、日本語文字などの列。「”（ダブルクォーテーション）」を直接文字定数の中に記述することはできない。

例　”ＡＢＣＤアイウエオ日本語”

整数型数値定数

・10進形式………－32768～＋32767までの整数。数の最後に「％（パーセント）」を付けて表わす。「．（小数点）」を付けることはできない。　例　8561、256％

・16進形式………先頭に「＆ｈ（アンパーサンド　エイチ）」を付けて表わす。　例　&h1F00

・8進形式………先頭に「＆ｏ（アンパーサンド　オー）」を付けて表わす。　例　&o17

実数型数値定数

・単精度実数………有効桁7桁で計算を行ない、出力されたときに7桁目を四捨五入して、6桁以下で表示される。扱える数の範囲は－1.70141E＋38～1.70141E＋38までである。指数形式は、「E」を使って表わす。数の最後に「！（イクスクラメーションマーク）」を付けて表わす。　例　149.3！

・倍精度実数………有効桁16桁で計算を行ない、16桁以下で表示される。指数形式は、「D」を使って表わす。数の最後に「＃（シャープ）」を付けて表わす。

例　6789756.4321＃

⑩変　数

●メモリ上にデータを格納する

一般に数値や文字列などのデータを入れる箱のようなものとして説明されますが、パソコンの中では、メモリ上にデータを格納する場所を変数名で指定して管理しています。

変数名

変数名には40文字までのアルファベットと数字、「．(ピリオド)」を使うことができますが、初めに必ずアルファベットがこなければなりません。

また、予約語(BASICで使用しているコマンドや命令の名まえなど）であってはいけません。しかし、変数名の中に予約語が含まれることは許されます。FNで始まる変数名もユーザー定義関数を示すので、付けることはできません。

変数の型は、定数と同じ記号を変数名の最後に付けて指定します。この場合、型が違うと同じつづりの変数でも違うものとして扱われます。

配列変数

1つの変数名に番号（配列要素の添え字）を付けて、たくさんのデータを変数に格納して取り扱います。添え字は0から始まり、最大値をDIM文で宣言して設定します。

なお、10までの添え字の配列変数は、宣言をしなくても使用することができます。

システム変数

BASICインタプリタのシステムが使用している変数で、ユーザーが自由に利用することのできない変数です。

システム変数には、TIME＄、DATE＄、ERR、ERL、MID＄があります。

⑪ 演算子

●計算や代入を行なう

式

式とは、定数や変数を演算子で結んで計算、あるいは代入を表わす一般的な数式だけでなく、単に文字や数値、変数だけで表わされる場合もあります。

例　123.4、”アイウエオ”、10＋3、INT (AA)

演算子

演算子には次のようなものがあります。

演算子	意　味	例
^	指数演算	10 ^ 4　(10^4)
−	負号	−120
*、／	乗算、除算	2＊4、9／3（2×4、9÷3）
＋、−	加算、減算	5＋4、6−2

演算の優先順序は指数演算から、加算減算までの上の並びの順になります。

また、「(　)（カッコ）」を使って演算の優先順位を指定するのは一般の計算と同様ですが、カッコ内では優先順位に従って計算されます。

¥（整数除算）

被除数を除数で割った商の小数点以下を切り捨てた値が結果となります。計算前に実数の小数点以下が四捨五入されます。

例　10.3　¥　3＝3…1　　結果　3

MOD（剰余計算）

被除数を除数で割った結果の余りを計算します。計算前に実数の小数点以下が四捨五入されます。

例　10.3　MOD　3＝3…1　結果　1

2. BASICプログラムの編集

❶プログラムの作成

●行番号を付けて1行ごとにメモリに登録

BASIC起動直後は、カーソルが点滅していてユーザーからコマンドが入力されるのを待っています。この状態を「コマンド待ちのレベル」などといいます。

この状態で、キーボードからコマンドや命令を入力し、⏎を押すと、ただちに命令が実行されます。これがダイレクトモードです。

プログラムを作成するときは、同じようにコマンド待ちの状態で、先頭に行番号としての整数と区切りのスペースをおいて、続けて命令を文として入力します。そして⏎キーを押すと、その行番号が付けられた1行分のプログラムがメモリ上に登録されます。

例　10　PRINT "ABCDEFG"⏎

コマンド待ちのレベルは、BASICでは「エディットモード」でもあります。エディットとは「編集作業」のことで、BASICプログラムを作ったり、修正したり、消去したりする作業のことです。

プログラムは、エディットモードで、行番号に続いて命令を書き並べて作成していきますが、プログラムが完成、あるいはその一部ができあがったときに、エディットモードのままでRUNコマンド、またはGOTO文で開始行番号を指定して、プログラムを走らせてみることができます。

こうして作成中、あるいは完成したときに、その場でプログラムを走らせてみて、エラーがでたり不つごうな点がでたら、それを訂正しながらプログラムを完成に近づけていくのです。

❷プログラムの表示

●LISTコマンドで表示させ修正や追加ができる

旧タイプのPC−98シリーズの本体付属の「N_{88}−日本語BASIC(86)」は、逐次命令を解釈しながらプログラムを実行するインタプリタ型言語のひとつです。

これに対して、コンパイラ型言語という言語もあります。これは、プログラムをエディタで作成して、一括して機械語に翻訳してから実行される言語です。

インタプリタ型言語には、プログラムを作成しながらテストランしたり、エラーが発見されたらすぐにプログラムの実行が停止して、プログラムを訂正できる、などの特徴があります。このため、インタプリタ型言語は学びやすく、使いやすいといわれています。

プログラムは、キーボードから入力したり、すでに作ってあるものはフロッピーディスクから読み込んで、パソコンのメモリ上に記憶させます。

ユーザーがプログラムを確認したり修正するために、プログラムの内容を見る必要がある場合、LISTコマンドでその内容をディスプレイに表示することができます。

LISTコマンドが実行されて、プログラムのリストが表示されると、必ずコマンド待ちのレベルに戻りますので、このディスプレイ上に表示されたプログラムをただちに修正したり、新たにプログラムを追加したりすることができます。

インタプリタ型言語とは、通訳型言語とでもいったらいいでしょうか。つまり、BASICは人間にわかりやすく作られたコンピュータ言語ですが、プログラムを実行するときには、このBASICで書かれた命令のひとつひとつを、コンピュータが直接実行できる機械語に通訳しながら実行していくのです。

❸プログラムの修正

●カーソルを修正箇所に重ねてキーボードから入力

プログラムは1行を単位として、自由に修正することができます。修正するためには、まずLISTコマンドで修正の必要な行をディスプレイ上に表示させます。そして、カーソルを修正する箇所に重ね合わせて、キーボードから入力して訂正します。

また、EDITコマンドを使って、プログラムをロールアップ・ロールダウンさせながら、修正することもできます。

プログラムのエディットに使われるキーは、次のとおりです。

エディットキー	役　割
ROLL UP	プログラムを巻物に見立てて、一部をディスプレイで見ながら上に巻き上げていくように表示する。
ROLL DOWN	ROLL UPの反対の動作。
INS	重ね合わされた文字を右に送りながら、キーボードから入力された文字を挿入していくモードにする。元に戻すには、⏎かもう一度INSキーを押す。
BS	カーソルのある左側の文字を消して1文字詰める。
DEL	カーソル位置の文字を消して、1文字詰める。
HOME	カーソルをホームポジション（左上隅）に移動する。
CLR	画面をぜんぶ消去する。
HELP	プログラムがエラーストップした場合、このキーを押すとエラーの発生したか所にカーソルをおいて、その行をリストアップする。
STOP	プログラムの実行を中断する。
CTRL +S	LISTコマンドでリスト表示中にこのキーを押すと、表示を一時停止する。

❹プログラムの追加

●追加、挿入のあとはRENUMで行番号を整理

追　加

BASICでは、プログラムは行番号を単位として管理されています。このため、現在のプログラムの最後の行番号より大きい行番号で1行のプログラムを登録すれば、プログラムの最後尾に1行プログラムを追加することができます。

これは、AUTOコマンドで自動的に実行することができます。AUTOコマンド実行中は、⏎キーが押されて1行分のプログラムが登録されるたびに、指定された行番号の増分に応じた行番号を発生して、次の文の入力待ちになります。こうすると、いちいち行番号を入力する手間が省けるわけです。

挿　入

また、行と行の間にプログラムを挿入する場合は、行間の番号でプログラムを登録することによって、新たにプログラムを挿入することができます。

このために、一般的にプログラムを10番おきの行番号で登録しています。こうするといつでも9行までの挿入ができます。

行番号の整理

行の追加や挿入が行なわれて、行番号がきちんと整列しなくなったりした場合は、RENUMコマンドで行番号をそろえます。

RENUMコマンドでは、GOTO文の飛び先や、IF ~ THEN ~ ELSE文の分岐先、ON　GOSUB文のサブルーチンの実行行番号なども自動的に付け替えられます。

⑤プログラムの登録

● SAVE コマンドでディスクに書き込む

キーボードから入力されたプログラムは、パソコン本体のメモリ上に登録されて、いつでも実行可能な状態ですが、電源スイッチをOFF にすると、すべて失われてしまいます。

パソコンのスイッチを入れるたびにプログラムをキーボードから入力していたのでは、とても実用になるものではありません。このために、多くの外部記憶装置が用意されています。

PC−9801 シリーズにも各種の外部記憶装置が用意されていますが、たいていの本体にはフロッピーディスク装置が内蔵されています。また、ハードディスクが内蔵されているものもあります。

本体に内蔵されているからといって、これらの記憶装置を「内部記憶装置」とはいいません。外部記憶装置とは、パソコンの頭脳である中央処理装置（CPU）からみて外部にあるということなのです。

プログラムをフロッピーディスクに登録するためには、SAVE コマンドで、登録すべきフロッピーディスクの入っているドライブ名（番号）と、プログラムの名まえを指定して登録します。

一度登録したプログラムは、KILL コマンドで消去されるまで残ります。

同じ名まえで登録されると、以前に登録されていたプログラムが消去され、新しいプログラムが登録されてしまいます。

プログラムがまちがって消去されたりしないように、SET 文で「読み出し専用」などの属性を設定することもできます。

また、登録されているプログラムの名まえを NAME コマンドで付け替えることもできます。

❻ プログラムの呼び出し

● LOAD コマンドでいつでも呼び出し実行

フロッピーディスクに登録されたプログラムは、LOADコマンドでいつでもパソコン本体に呼び出して、実行したり、編集したりすることができます。

フロッピーディスク上のプログラムは、呼び出すべきプログラムの登録されているディスクの入っているフロッピーディスクのドライブ名（番号）と、プログラムの名まえを指定して呼び出します。

呼び出したプログラムは、キーボードから入力したプログラムとまったく同様に扱うことができます。

プログラムを実行するためにフロッピーディスクから呼び出すのであれば、RUN コマンドに続けてドライブ名（番号）と、プログラムの名まえを指定して呼び出すことによって、本体内にプログラムが読み込まれるとすぐにそのプログラムの実行を開始します。

また、BASIC 起動直後に自動的にプログラムを読み込んで実行させることもできます。これはオートスタートといって、本体付属のシステムディスクにある「setinf. n88」というユーティリティプログラムを使います。詳しくは本体に付属してくるBASICユーザーズマニュアルを参照してください。

そのほかに、CHAIN 文を使って他のプログラムからフロッピーディスク上のプログラムを呼び出して、実行を移すこともできます。

PC-98シリーズ BASIC 用語・用例辞典



1. 基本コマンド

AUTO（オート）

■行番号を自動的に付ける

書式　AUTO　開始行番号，行番号増分

➔プログラムを作成するときに、指定した「開始行番号」から自動的に行番号を付ける。

➔⏎（リターンキー）で1行を確定するたびに、「行番号増分」ずつ増加した行番号を自動的に発生する。

➔STOP（ストップキー）を押すと、行番号の発生を中止して、BASICのコマンドレベルに戻る。

➔プログラム中にすでに存在する行番号が発生したときは、行番号の後ろに「*（アスタリスク）」が付いて、注意を促す。

➔文字を入力して⏎キーを押すと、その内容が登録され、内容を変更することができる。ただし、⏎キーだけを押すと、行が削除され内容が失われるので、注意する。

➔「開始行番号」、「行番号増分」には、それぞれ10が省略値として設定されている。したがって、「開始行番号」「行番号増分」を省略すると、それぞれこの省略値を指定したことになる。

用例

AUTO　100 ⏎

100行から10ずつの増分で行番号を発生させる。

AUTO　1000, 20 ⏎

1000行から20ずつの増分で行番号を発生させる。

EDIT（エディット）

■画面上で効率的にプログラムを編集する

書式　①　EDIT　行番号
②　EDIT.（ピリオド＝小数点）

➔画面上で効率的にプログラムを編集するために、指定された行番号のプログラムを表示してカーソルを行の先頭に置く。
➔画面を［ROLL UP］・［ROLL DOWN］キーで上下にスクロールさせながら、プログラムの編集をすることができる。
➔書式②の「．（ピリオド＝小数点）」は、現在実行している行番号か、直前にリターンキーを押してプログラムに登録した行番号を示している。これがBASICが管理しているポインタで、実行中断された位置や最後に修正された位置なども記憶している。書式②では、このポインタのある行を表示して、カーソルを先頭に置く。
➔プログラムがエラーストップした場合などは、書式②でエラーストップした行を表示して、訂正することができる。

用例

EDIT　200　⏎

200行を表示してカーソルを先頭に置き、プログラムを修正する準備をする。

EDIT .　⏎

現在ポインタのある行を表示して、カーソルを先頭に置き、プログラムを修正する準備をする。

LIST／LLIST（リスト／エルリスト）

■プログラムをディスプレイに表示／印刷する

書式
① LIST
② LIST　行番号
③ LIST　開始行番号－終了行番号
④ LIST　開始行番号－
⑤ LIST　－終了行番号
⑥ LIST　.
⑦ LLIST

➔メモリ上にあるプログラムの内容をディスプレイに表示、またはプリンタから印刷する。

➔書式①は、プログラムの初めから表示を開始し、プログラムの最後までを表示する。

➔書式②は、「行番号」で指定した行だけを表示する。

➔書式③は、「開始行番号」で指定した行から「終了行番号」で指定した行までを表示する。範囲は、「－（マイナス）」記号ではさんで指定する。

➔書式④は、「終了行番号」を省略して「開始行番号」と「－（マイナス）」記号だけを指定すると、「開始行番号」から、プログラムの最後までを表示する。

➔書式⑤は、「開始行番号」を省略して「－（マイナス）」記号と「終了行番号」だけを指定すると、プログラムの初めから「終了行番号」までを表示する。

➔書式⑥は、「.（ピリオド）」を指定すると、BASICインタプリタが管理しているポインタ（実行中断された位置や、最後に修正された位置なども記憶している）の示す行を指定したことになる。

➔書式⑦は、プリンタにリストを印刷する。リストを印刷する範囲は、LIST文と同様に指定することができる。

➔ CTRL ＋S キーを押すと、リスト表示を一時停止する。

➔ STOP キーを押すことにより、リスト表示または印刷を中断することができる。

用例

LIST ⏎	………プログラムのリストをぜんぶ表示する。
LIST 100 ⏎	………100行のリストを表示する。
LIST 100－300 ⏎	………100行から300行までのリストを表示する。
LIST 500－ ⏎	………500行から最後までのリストを表示する。
LIST －500 ⏎	………プログラムの最初から500行までのリストを表示する。
LIST ⏎	………ポインタの示す行のリストを表示する。
LLIST ⏎	………プログラムのリストをぜんぶ印刷する。
LLIST 100 ⏎	………100行のリストを印刷する。
LLIST 100－300 ⏎	……100行から300行までのリストを印刷する。
LLIST 500－ ⏎	………500行から最後までのリストを印刷する。
LLIST －500 ⏎	………プログラムの最初から500行までのリストを印刷する。
LLIST．． ⏎	………ポインタの示す行のリストを印刷する。

DELETE（デリート）

■プログラムの一部を削除する

書式　DELETE　開始行番号－終了行番号

➔指定した「開始行番号」から「終了行番号」までのプログラムを削除する。

➔「開始行番号」だけを指定した場合は、その行だけが削除される。

➔「－（マイナス）」と「終了行番号」のみを指定した場合は、プログラムの先頭から「終了行番号」までを削除する。

➔「開始行番号」と「－」だけで、「開始行番号」以降の行すべてを削除するという指定はできない。このような場合は、プログラムの最終行番号を「終了行番号」として指定する必要がある。

用例

DELETE　500 ⏎

500行をプログラムから削除する。

DELETE　500－900 ⏎

500行から900行をプログラムから削除する。

DELETE　－500 ⏎

先頭の行から500行までをプログラムから削除する。

RENUM（リナンバー）

■プログラムの行番号を新しく付けなおす

書式　RENUM　新行番号，現行番号，行番号増分

➔「新行番号」は新しく付ける行番号の最初の行番号を指定する。

➔「現行番号」は付け替えを始めるプログラムの、現在の行番号を指定する。

➔「行番号増分」は新しく付ける行番号の間の増分を指定する。

➔「新行番号」、「現行番号」、「行番号増分」の省略値は10で、指定を省略した場合は10を指定したことになる。

➔プログラム中、GOTO、GOSUB、IF～THEN、ON GOTO、ON GOSUB などで指定している飛び先の行番号も、自動的に新しい行番号に対応して付けなおされる。

➔もし、これらの指定している飛び先の行番号が存在しない場合は、「Undefined line 誤番号 in 新行番号」とエラーメッセージが表示される。

これは、付けなおされた「新行番号」に対応する行番号がないため、付けなおすことができなかった「誤番号」が存在することを示している。

用例

RENUM　100, 91, 10 ⏎

現在のプログラムの91行から最終行までを、新しく100行から10ずつの増分で付けなおす。

LOAD（ロード）

■ディスクからプログラムをメモリに読み込む

書式　① LOAD　"ドライブ名：ファイル名"
　　　　② LOAD　"ドライブ名：ファイル名", R

➔書式①は、「ドライブ名」で指定されたディスクドライブのフロッピーディスクから、「ファイル名」で指定されたプログラムを本体のメモリ上に読み込む。

➔書式②は、「ドライブ名」で指定されたディスクドライブのフロッピーディスクから、「ファイル名」で指定されたプログラムを本体のメモリ上に読み込み、ただちにこのプログラムを実行する。

用例

LOAD "2：PROG1．BAS" ⏎

ドライブ2から「PROG1．BAS」というプログラムを本体メモリ上に読み込む。

LOAD "2：PROG2．BAS", R ⏎

ドライブ2から「PROG2．BAS」というプログラムを本体メモリ上に読み込み、ただちに実行する。

SAVE（セーブ）

■メモリ上のプログラムをディスクに登録する

書式
① SAVE ”ドライブ名：ファイル名”
② SAVE ”ドライブ名：ファイル名”，A
③ SAVE ”ドライブ名：ファイル名”，P

➔書式①は、本体のメモリ上にあるBASICプログラムを、「ドライブ名」で指定されたディスクドライブのフロッピーディスクに、指定された「ファイル名」で書き込む。

➔書式②は、本体のメモリ上にあるBASICプログラムを、「ドライブ名」で指定されたディスクドライブのフロッピーディスクに、指定された「ファイル名」で、アスキー形式で書き込む。

➔書式③は、①と同じようにフロッピーディスクに書き込むが、LISTやEDITで内容を見たり、変更したりすることはできない。

➔指定された「ファイル名」がすでにディスク上に登録されている場合は、古いファイルは失われて、新しいファイルに更新される。

用例

SAVE ”2：PRGRM1．BAS” ⏎

本体メモリ上のプログラムを、「PRGRM1．BAS」というファイル名で、ドライブ2のフロッピーディスクに登録する。

SAVE ”2：PRGRM2．BAS”，A ⏎

本体メモリ上のプログラムを、「PRGRM2．BAS」というファイル名で、ドライブ2のフロッピーディスクに、アスキー形式で登録する。

注 アスキー形式：ふつうのプログラムファイルは、バイナリ形式という圧縮されたデータ（メモリに格納されているときのイメージ）

としてフロッピーディスクに登録されるが、アスキー形式では、データは圧縮されずに、LIST 命令で表示させたときのような文字列のイメージで登録される。

MERGE（マージ）

■メモリ上とディスク上のプログラムを合成する

書式　MERGE　"ドライブ名：ファイル名"

➔「ドライブ名」で指定されたディスクドライブのフロッピーディスクから、「ファイル名」で指定されたプログラムを読み込み、本体メモリ上のプログラムに合成する。

➔指定されたファイルは、アスキー形式でセーブされていなければならない。

➔メモリ上のプログラムとファイルのプログラムに同じ行番号があった場合は、読み込まれたファイルのプログラムの行が優先されて置き替えられる。

プログラムは、行番号が重ならないようにMERGE コマンド実行前に行番号を整理しておく必要がある。

用例

MERGE　"2：PRGRM1　.　BAS"　⏎

本体メモリ上のプログラムに、ドライブ2のフロッピーディスクにある「PRGRM1　.　BAS」というプログラムを合成して、1つのプログラムにする。

参照　SAVE／LOAD

FILES／LFILES
（ファイルス／エルファイルス）

■ファイルの名称、種類、大きさを出力する

書式 ① FILES ドライブ名
② LFILES ドライブ名

➔書式①は、「ドライブ名」で指定されたフロッピーディスク上に登録されているファイルの名称、種類、大きさをディスプレイに表示する。

➔書式②は、「ドライブ名」で指定されたフロッピーディスク上に登録されているファイルの名称、種類、大きさをプリンタに印字する。

➔「ドライブ名」の省略値は1で、省略した場合はドライブ1が選択されたことになる。

➔ファイル名の拡張子との区切り記号は、次の3つのファイルの種類を示している。

「　(空白)」……………………アスキー形式のファイルか、BASICのデータファイル

「．(ピリオド)」……………バイナリ形式のファイル

「＊(アスタリスク)」………マシン語のファイル

➔ファイルの大きさはクラスタ単位で示される。

用例

FILES　1 ⏎

ドライブ1のフロッピーディスクに登録されているファイルをディスプレイにリストアップする。

LFILES　2 ⏎

ドライブ2のフロッピーディスクに登録されているファイルをプリンタから印字、リストアップする。

NAME（ネーム）

■ディスク上のファイルの名称を変更する

書式

① NAME ”現ファイル名” AS ”新ファイル名”

② NAME ”ドライブ名：現ファイル名” AS ”ドライブ名：新ファイル名”

➔書式①は、すでに「現ファイル名」でフロッピーディスク上に登録されているファイルを「新ファイル名」に付け替える。

➔書式②は、「ドライブ名」も各ファイル名の前に指定することができるが、指定する「ドライブ名」は「現ファイル名」「新ファイル名」ともに同じでなければならない。

➔「ドライブ名」の省略値は1で、省略した場合はドライブ1が選択されたことになる。

用例

NAME ”PRGRM1 . BAS” AS ”HENKOU . BAS” ⏎

ドライブ1のフロッピーディスクに登録された「PRGRM1 . BAS」のファイル名を「HENKOU . BAS」に変更する。

NAME ”2：PRGRM2 . BAS” AS ”2：SINMEI . BAS” ⏎

ドライブ2のフロッピーディスクに登録された「PRGRM2 . BAS」のファイル名を「SINMEI . BAS」に変更する。

KILL（キル）

■ディスクに登録されているファイルを削除する

書式　① KILL "ドライブ名：ファイル名"

➔「ドライブ名」、「ファイル名」で指定されたフロッピーディスク上に登録されているファイルを、フロッピーディスクから削除する。

➔フロッピーディスク上のすべてのファイルを削除することができるが、一度に1つのファイルしか削除できない。

➔オープンされているファイルは削除できない。

➔削除されたファイルは元に戻すことはできないので、KILL 命令は慎重に使わなければならない。

用例

KILL "PRGRM1．BAS" ⏎

　ドライブ1のフロッピーディスク上に登録されている「PRGRM1．BAS」というファイルをフロッピーディスク上から削除する。

KILL "2：JUSHO．DAT" ⏎

　ドライブ2のフロッピーディスク上に登録されている「JUSHO．DAT」という BASIC のデータファイルをフロッピーディスク上から削除する。

SET（セット）

■ファイル属性を設定、解除する

書式　① SET　ドライブ名 ,"属性文字"
② SET "ドライブ名：ファイル名","属性文字"
③ SET #ファイル番号 ,"属性文字"

➔設定できる属性は次の「属性文字」で表わされる2種類で、
「P：ライトプロテクト」＝書き込みの禁止
「R：リードアフターライト」＝書き込みの確認
を示している。

➔「P属性」は、同一ファイル名での追加登録ができなくなり、KILLによるファイルの削除もできなくなる。また、PRINT #、PUT #、WRITE #などによるファイルへの書き込みが禁止される。

➔「R属性」は、ファイルの書き込みの後、読み込みを行ない、正しく書き込みが行なわれたか確認する。

➔「属性文字」としてP、R以外の文字を指定するとファイル属性が解除される。

用例

SET 1,"R" ⏎

ドライブ1全体にR属性を設定する。

SET "2：PRGRM1．BAS","P" ⏎

ドライブ2のフロッピーディスク上のファイル「PRGRM1．BAS」にP属性を設定する。

SET "2：PRGRM1．BAS","" ⏎

ドライブ2のフロッピーディスク上のファイル「PRGRM1．BAS」のP属性を解除する。

RUN（ラン）

■プログラムを実行する

書式　① RUN
② RUN　行番号
③ RUN　"ドライブ名：ファイル名"
④ RUN　"ドライブ名：ファイル名"，R

➔メモリ上にあるプログラムを実行する。
➔書式①は、プログラムの先頭から実行する。
➔書式②は、指定した「行番号」からプログラムを実行する。
➔書式③は、指定したドライブのフロッピーディスクから、「ファイル名」で指定したプログラムをロードして実行する。このとき、すべてのオープンされているファイルをクローズし、メモリの内容をクリアする。
➔書式④は、すでにオープンされているデータファイルをそのままのオープン状態で、指定したドライブのフロッピーディスクから「ファイル名」のプログラムをロードして実行する。

用例

RUN ⏎

メモリ上にあるプログラムの先頭から実行する。

RUN "2：PRGRM1．BAS" ⏎

ドライブ2のフロッピーディスク上に登録された「PRGRM1．BAS」というプログラムをロードして実行する。

CONT【コンティニュー】

■ 停止したプログラムの実行を再開する

書式　CONT

➔ [STOP] キーを押したり、[CTRL]＋C キー（[CTRL] キーを押しながらC キーを押す）でプログラムの実行を停止した場合、CONT コマンドでプログラムの実行を再開する。

➔プログラムの中にSTOP文を入れ、一定のところでプログラムを停止させ、CONT コマンドで実行を再開する。

➔ CONT コマンドは、ふつうデバッグ（プログラムの誤りを訂正すること）作業のために使用される。プログラムを一時停止し、ダイレクトモードで変数の内容やディスプレイ画面などの実行状態を調べ、CONT コマンドで実行を再開する。

➔プログラムの実行が停止されているときにプログラムの内容を変更すると、CONT コマンドによる実行の再開ができなくなるので、注意する。このような場合は、RUN コマンド、または GOTO 文で初めから実行しなおすか、または指定した行番号から実行を開始する。

用例

① [STOP] キーでプログラムの実行を停止する。
② PRINT A, B$ [↵] などのように変数の内容を調べる。
③ CONT [↵]
　　プログラムの実行を再開する。

TRONノTROFF
（トレースオン／トレースオフ）

■プログラムの実行状態を表示する

書式　① TRON
　　　② TROFF

➔書式①を実行すると、それ以降プログラムの実行にともなって、実行するプログラムの行番号をディスプレイ画面に表示するトレースオンの状態となる。

➔表示する行番号は、実行中のプログラムと区別するため「[ ]（かぎかっこ）」で囲まれて表示される。

➔書式②を実行すると、トレースオンの状態が解除され、実行中の行番号を表示しない通常の状態に戻る。

➔TRONは、プログラムのデバッグ（プログラムの誤りを訂正すること）作業のため、プログラムの実行状態を監視して、実行の流れを追跡するのに使用される。

➔トレースオンの状態はNEWコマンドを実行することによっても解除される。

用例

TRON ⏎

　　プログラムの実行状態を追跡する。

TROFF ⏎

　　プログラムの実行状態を追跡するトレースオンの状態を解除する。

NEW（ニュー）

メモリ上のプログラムを消去し変数をクリアする

書式　NEW

➔本体のメモリ上にあるプログラムを消去し、使用されたすべての変数をクリアして、数値変数は0、文字列変数はヌル（空＝カラ）とする。

➔一度NEWコマンドによって消去されたプログラムは元に戻すことができないので、NEWコマンドは慎重に使わなければならない。

➔NEWコマンドを実行すると、オープンされているファイルがある場合は、自動的にすべてクローズする。

用例

NEW ⏎

現在本体のメモリ上にあるプログラムを消去して、新たなプログラムを作成する準備をする。

NEW ON（ニュー オン）

■BASICのシステムを再起動する

書式 NEW ON 機能番号

➔「機能番号」で指定したシステムを再起動する。

0………N88-BASICを起動

1………N-BASICを起動

2………ターミナルモードで起動

➔NEW ONを実行するときは、再起動するBASICのシステムディスクをフロッピーディスクに入れておかなければならない。

➔ターミナルモードのとき、各種通信モードの設定は本体のディップスイッチで行なうため、NEW ON文では設定できない。設定するためにはTERM文を使わなければならないので注意する。

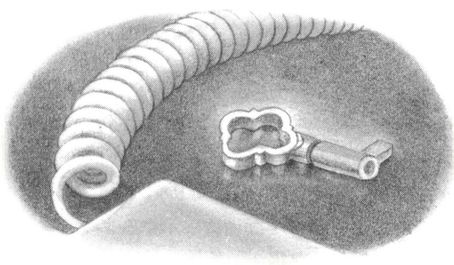
用例

NEW ON 0 ⏎

N88-BASICを再起動する。

NEW ON 1 ⏎

N-BASICを再起動する。



2. 基本入出力命令

PRINT／LPRINT（プリント／エルプリント）

文字列や変数の内容を表示／印字する

書式　① PRINT　データ　区切り記号　データ
　　　② LPRINT　データ　区切り記号　データ

➔書式①は、指定された「データ」（文字列や変数の内容）をディスプレイに表示して改行する。

➔書式②は、指定した「データ」（文字列や変数の内容）をプリンタに印字する。他の働きはPRINT文と同じである。

➔「データ」などを省略すると改行のみを行なう。

➔省略型として書式①は「？（クエスチョンマーク）」、書式②は「L？」を使用することができる。

➔「データ」は、変数を指定するとその内容を表示する。文字列は、「"（ダブルクォーテーション）」で囲んで指定するとそのまま表示される。計算式を指定すると、計算した結果の値が表示される。

➔「区切り記号」は、「；（セミコロン）」および「，（カンマ）」を使うことができる。

➔「；（セミコロン）」は、「データ」を続けて表示する。「，（カンマ）」は、あらかじめ14文字ごとに区切られた表示領域の次の領域の先頭から続いて表示される。

➔PRINT文の最後に「区切り記号」を置くと、次のPRINT文が実行した結果を続けて画面に表示するように指定することができる。

➔数値を表示した場合、数値の最後にスペースが付加される。また、

正の数値を出力した場合には、符号が付加されるためのスペースが数字の先頭に1つ入れられるので、連続して出力する場合には注意する。

用　例

```
100 A=12345
110 B=-1234
120 A$="ABCDEF"
130 B$="漢字かな"
140 PRINT "0123456789012345678900123456789"
150 PRINT "ABCDEF"
160 PRINT A
170 PRINT B
180 PRINT A$
190 PRINT B$
200 PRINT
210 PRINT A;B;A$;B$;123;-123;"ABCDEF"
220 PRINT A,B,A$,B$,123,-123,"ABCDEF"
230 PRINT "0123456789012345678900123456789"
```

参　照　PRINT USING／LPRINT USING／TAB／WRITE

●PRINT　**応用プログラム**

アニメーションクルーザーを動かしてみよう

```
100 '************************************
110 '豪華クルーザーを動かそう
120 '************************************
130 CONSOLE 0,25,0,1
140 WIDTH 80,25
150 CLS
160 SP1$="       >          "
170 SP2$="  ~    XOLE       "
180 SP3$=" --____JDDDD____/-"
190 SP4$=">>>L___________/"
200 COLOR 5
210 LOCATE 0,18:PRINT STRING$(80,"-")
220 COLOR 6
```

スペース（160行の先頭部分）
クルーザーを文字列で変数に代入（160〜190行）
波の部分（190行の">>>"）
水平線を表示（210行）

```
230 FOR I=1 TO 4000:NEXT I
240 FOR X=0 TO 63
250   LOCATE X,15:PRINT SP1$;
260   LOCATE X,16:PRINT SP2$;
270   LOCATE X,17:PRINT SP3$;
280   LOCATE X,18:PRINT SP4$;
290   FOR T=1 TO 100:NEXT T
300 NEXT X
310 COLOR 7
320 END
```

（240〜300行）表示位置を変える

（250〜280行）クルーザーを表示

●表示させたり消したりの繰り返し

PRINT文を応用するだけで、簡単なアニメーションを作ることができる。

プログラムをRUNすると、クルーザーがディスプレイ画面の左から右へ疾走する。

160〜180行のクルーザーの絵の左端のスペースで、直前に描いた絵を消している。

190行の波の部分は、スペースがないので消えずに残る。

WRITE（ライト）

■文字列や変数の内容をディスプレイに表示する

書式　WRITE　データ　区切り記号　データ

➔指定した「データ」を表示して改行する。

➔「データ」は、変数を指定するとその内容を表示する。文字列は、「”（ダブルクォーテーション）」で囲んで指定すると、そのまま表示される。計算式を指定すると、計算した結果の値が表示される。

➔「データ」などを省略してWRITE文だけを実行するとエラーとなる。

➔「区切り記号」は、「；（セミコロン）」および「，（カンマ）」を使うことができるが、働きの区別はなく、データの区切りとして必ず「，（カンマ）」を表示する。

➔ PRINT 文と同じような画面への出力機能を持っているが、データ間の不用な空白は詰められ、各データ間は必ず「，（カンマ）」で区切られて出力される。また、文字列は「 ”（ダブルクォーテーション）」で囲まれて出力される。

➔出力するデータをなるべく詰めて表示したい場合などに WRITE 文を使用する。しかし、一般的にはおもに PRINT 文を使い、WRITE 文は画面への表示を目的としては、あまり使われない。

むしろ、データをフロッピーディスクなどに記録するときに、記録スペースの節約を目的として「WRITE #」文として使われることが多い。この命令は、データ書き込みのイメージを画面上で確認するために役立つ。

用 例

```
100 A=12345
110 B=-1234
120 A$="ABCDEF"
130 B$="漢字かな"
140 PRINT "0123456789012345678900123456789"
150 PRINT "ABCDEF"
160 WRITE "ABCDEF"
170 PRINT A
180 WRITE A
190 WRITE B
200 WRITE A$
210 WRITE B$
220 PRINT
230 WRITE A;B;A$;B$;123;-123;"ABCDEF"
240 WRITE A,B,A$,B$,123,-123,"ABCDEF"
250 PRINT "0123456789012345678900123456789"
```

参 照　PRINT／WRITE #

PRINT USING（プリント ユージング）

LPRINT USING（エルプリント ユージング）

■文字列、数値、変数などを編集して出力する

書式　① PRINT USING ”書式制御文字列”；データ
　　　② LPRINT USING ”書式制御文字列”；データ

➔書式①はディスプレイに、書式②はプリンタに、それぞれ「データ」を「書式制御文字列」で指定した書式に編集して表示、印字する。

➔日本語を含む文字列は編集することはできない。

➔「書式制御文字列」は、指定した数値の出力範囲を超えた数値を出力する場合、または数値が出力範囲を超えて四捨五入などの処理が行なわれた場合などは、数値の先頭に「%（パーセント）」が付加されて注意を促す。

➔おもな「書式制御文字列」には以下のようなものがある。

「#（ナンバー）」………出力する数値の桁数を指定して、右詰めで表示する。

「．（ピリオド）」………小数点の位置を指定し、小数点以下がない場合は0を出力する。

「＋（プラス）」………「書式制御文字列」の前後に付けた場合、それぞれ数値の前、または後ろに「＋、－」の符号を付加する。

「¥¥(円マーク)」….「書式制御数字列」の先頭に付けた場合、数値の頭に「¥」を付加する。

「，（カンマ)」……… 桁数指定の「#」の中に置いた場合、数値の整数部が3桁ごとに「，(カンマ)」で

区切られて出力される。

「&＿＿&（アンパーサンドにはさまれたスペース）」………

文字列や文字変数の内容から、スペースの数＋アンパーサンドの数分の文字列を先頭から切りとって、左詰めで表示する。

用例

```
100 A=12345
110 B=-12345
120 A$="ABCDEF"
130 PRINT "01234567890123456789001234567 89"
140 PRINT USING "##########";A
150 PRINT USING "##########";B
160 PRINT USING "######.###";A
170 PRINT USING "+#########";A
180 PRINT USING "+#########";B
190 PRINT USING "¥¥########";A
200 PRINT USING "######,###";A
210 PRINT USING "######,###";12345678#
220 PRINT USING "&         &";A$
230 PRINT "0123456789012345678900123456789"
```

参照 PRINT／LPRINT

INPUT（インプット）

■キーボードからデータを入力する

書式　INPUT　”プロンプト文”　区切り記号　変数，変数，
………

➔ INPUT 文が実行されると、データの入力待ちの状態になる。データはキーボードから入力され、キーを押すことによって指定された「変数」に代入される。

➔「プロンプト文」は、入力を促すため画面に表示するメッセージで、文字列を自由に設定することができる。

➔「区切り記号」は、「；（セミコロン）」および「，（カンマ）」を使うことができる。

➔「；（セミコロン）」は、「プロンプト文」の後に続いて「？（クエスチョンマーク）」と1文字分のスペースを表示し、カーソルが点滅して入力待ちになる。

➔「，（カンマ）」は、「プロンプト文」の後にはなにも表示されず、カーソルが点滅して入力待ちになる。

➔入力される変数は「，（カンマ）」で区切って複数指定できるが、データの入力の際にも同じ数だけ「，」で区切ってデータを入力しなければならないので、一般には複数を指定しない。

➔入力されるデータの型と、代入される変数の型は一致していなければならない。

➔文字列の代入には、「”（ダブルクォーテーション）」を省略できるが、「，（カンマ）」や文字列の前後に「　（スペース）」を置く場合は、「”（ダブルクォーテーション）」で囲わなければならない。

➔変数の型や個数が違っていた場合、「? Redo From start」と画面に表示して、再度データの入力待ちになる。このため画面が乱れる

ので注意する。

➔なにも入力せずに⏎キーだけを押すと、数値変数には0、文字変数にはヌルストリング（空＝カラの文字列）が入力される。

用　例

```
100 INPUT A
110 PRINT A
120 INPUT A,B
130 PRINT A,B
140 INPUT A$
150 PRINT A$
160 INPUT "スウチ A";A
170 INPUT "スウチ B";B
180 PRINT "スウチ A =";A,"スウチ B ="B
190 INPUT "スウチ A",A
200 INPUT "スウチ B";B
210 PRINT "スウチ A =";A,"スウチ B ="B
220 INPUT "名前";NAMAE$
230 PRINT "こんにちは ";NAMAE$;" 様"
```

参　照　　LINE　INPUT／INPUT　WAIT

LINE　INPUT（ライン　インプット）

■キーボードから1行分の文字列データを入力

書　式　　LINE INPUT　"プロンプト文"；文字変数

➔LINE　INPUT 文が実行されると、データの入力待ちの状態になる。データはキーボードから入力され、⏎キーを押すことによって指定された「文字変数」に代入される。

➔「プロンプト文」は、入力を促すため画面に表示するメッセージで、文字列を自由に設定することができる。

➔「プロンプト文」の後に「？（クエスチョンマーク）」などは表示されない。

➔入力されるデータは、255文字以内の文字列に限られ、「，（カンマ）」などの区切り記号も含めて文字変数に代入される。

用例

```
100 LINE INPUT A$
110 PRINT A$
120 LINE INPUT "PLESE INPUT B$ = ";B$
130 PRINT B$
140 LINE INPUT "「，（カンマ）」を含んだ文字列を入力してください？";C$
 150 PRINT C$
```

参照　INPUT／LINE INPUT WAIT

INPUT WAIT（インプット　ウェイト）

LINE INPUT WAIT（ライン インプット ウェイト）

■待ち時間を制限してデータを入力する

書式　① INPUT WAIT 待ち時間 ”プロンプト文” 区切り記号 変数，変数，………

② LINE INPUT WAIT 待ち時間 ”プロンプト文”；文字変数

➔待ち時間で指定された時間だけキーボードからの入力を待つほかは、書式①はINPUT文と、書式②はLINE INPUT文と機能は変わらない。待ち時間は、0.1秒単位で指定する（10の指定で1秒間の待ち時間）。

➔マルチステートメント（1行に複数のステートメントを記述すること）で、この文の後に他の文を続けた場合、待ち時間の制限以内に入力が行なわれた場合にのみ後続の文を実行する。

➔待ち時間の制限以内に入力が行なわれなかった場合は、後続の文は実行されず、次の行に実行が移るので、マルチステートメントでプログラミングする際は注意する。

用 例

```
100 INPUT WAIT 100,A
110 PRINT A
120 LINE INPUT WAIT 100,"PLESE INPUT B$ = ";B$
130 PRINT B$
140 INPUT WAIT 200,"PLESE INPUT C,D = ";C,D
150 PRINT "C=";C,"D=";D
160 LINE INPUT WAIT 300,"「，（カンマ）」を含んだ文字列を入力してください？";E$
170 PRINT E$
180 LINE INPUT WAIT 100,"PLEASE INPUT F$";F$:
 F$="時間内にF$が入力されました。内容は ";
 F$;"です。":GOTO 200
190 PRINT "時間内にF$には何も入力されません
 でした。"
200 PRINT "終了にはRETキーを押してください。"
210 GOTO 180
```

参 照　INPUT／LINE INPUT

KINPUT（漢字インプット）

■キーボードから日本語文字列データを入力する

書式　KINPUT　文字変数

➔ KINPUT 文が実行されると、自動的に日本語入力モードに入り、「？（クエスチョンマーク）」を表示してキーボートからの日本語文字列データの入力待ち状態になる。データは⏎キーを押すことによって入力され、指定された変数に代入される。

➔ データが代入される文字変数は1つしか指定できない。また、プロンプト文も指定できない。

➔ KINPUT 文では、日本語文字列以外の文字を入力することはできない。

➔ KINPUT 文での入力の始まる位置は、偶数桁のみになる。

LOCATE 文などで奇数桁を指定すると、桁数＋1の桁から入力が始まる。

用例

```
100 KINPUT A$
110 PRINT B$
120 INPUT "ROMAJI DE NYUURYOKU SITEKUDASAI =";B$
130 PRINT "日本語文字列を入力してください  =";
140 KINPUT C$
150 PRINT C$
160 PRINT "012345678901234567890123456789O"
170 PRINT "奇数桁から入力できません=";
180 KINPUT E$
190 PRINT E$
```

参照　INPUT／LINE INPUT／LOCATE

READ～DATA（リード～データ）

■データを読み込み順番に変数に代入する

書式　READ　変数，変数，変数，………

　　～

　　DATA　データ，データ，データ，………

➔ DATA 文で定義されたデータを、先頭から順番に、READ 文で指定された変数に読み込んでいく。

➔読み込むデータと変数の型は一致していなければならない。

➔ DATA 文は、非実行文（REM 文同様プログラム実行の際に無視されて他に影響しない）で、プログラムのどの位置に置いてもかまわないが、READ 文は DATA 文のデータのうちまだ読み込んでいないデータから順番に読み込むので、データの順番と数には注意しなければならない。

➔ DATA 文には1行（255文字）に入るだけのデータを、1データごとに「，（カンマ）」で区切って定義できる。

➔文字列データの場合、その先頭または最後が「，（カンマ）」あるいは、「．（ピリオド）」のとき、または「　（スペース）」を置くときは、「”（ダブルクォーテーション）」で囲まなければならない。

➔ READ 文で読み込まれるデータの数が不足して読み込むべきデータがない場合、エラーとなる。また、データの数が多く、読み込まれないデータがあっても無視される。

➔データの読み込みを始める位置を指定するには、RESTORE 文を使用する。

用例

```
100 READ A,B,C,D$,E$,F$
110 DATA 1,2,3,ICHI-BAN,ﾆﾊﾞﾝ,三番
```

```
120 PRINT A,B,C
130 PRINT D$,E$,F$
140
150 READ G$,H$,I$
160 DATA ",「カンマ デ カコマレ」",".「ピ リオド」"," 「スペース」 "
170 PRINT"0123456789012345678901234567890"
180 PRINT G$;H$;I$
190 '
200 FOR I=1 TO 10
210   READ A$(I)
220 NEXT I
230 FOR I=1 TO 10
240   PRINT A$(I)
250 NEXT I
260 '
270 DATA AKIKO,ERIKO,KEIKO,SANAE,SHIZUKO,TOSHIKO,
NANAE,HIROMI,MARIKO,YASUKO
```

参照　RESTORE

RESTORE（レストア）

■ READ 文を読み込むデータ文の位置を指定する

書式　① RESTORE

② RESTORE　行番号

➔書式①は、次にくる READ 文が読み込む DATA 文の位置を、プログラム中の最初の DATA 文に指定する（最初のデータから読み込みを始める）。

➔書式②は、次にくる READ 文が読み込む DATA 文の位置を、指定した行番号の DATA 文に指定する（任意の位置のデータから読み込みを始める）。

➔指定した位置の DATA 文に複数のデータがある場合でも、行の先頭のデータから順番に読み込まれ、行の途中からの指定はできない。

用例

```
100 RESTORE 1010
110 READ A,B,C,D$,E$,F$
120 PRINT A,B,C
130 PRINT D$,E$,F$
140 '
150 RESTORE 1020
160 READ G$,H$,I$
170 PRINT"012345678901234567890123456789O"
180 PRINT G$;H$;I$
190 '
200 RESTORE 1000
210 FOR I=1 TO 10
220   READ A$(I)
230 NEXT I
240 FOR I=1 TO 10
250   PRINT A$(I)
260 NEXT I
270 '
1000 DATA AKIKO,ERIKO,KEIKO,SANAE,SHIZUKO,TOSHIKO,
NANAE,HIROMI,MARIKO,YASUKO
 1010 DATA 1,2,3,ICHI-BAN,ﾆﾊﾞﾝ,三番
 1020 DATA ",｢ｶﾝﾏ ﾃﾞ ﾊｼﾞﾏﾙ｣",".｢ﾋﾟﾘｵﾄﾞ｣"," ｢ｽﾍﾟｰｽ｣"
```

参照　READ~DATA



INKEY$ (インキーダラー)

■キーボードから1文字のデータを入力する

書式　INKEY$

➔現在のキーボードの状態を調べて、キーが押されていればその文字1文字分を、キーが押されていなければヌルストリング(空＝カラの文字) を返す関数。

➔押されたキーはディスプレイには表示されない。

➔[CTRL]＋Cキーおよび[STOP]キーは、INKEY$関数では拾うことはできない。

➔ふつう、用例のように、キーボードでキーが押されるまでループを繰り返して待機し、押されたキーを変数に代入して、その代入された文字を使って処理を続けるというようなプログラミングで使われる。

用例

```
100 PRINT"なにかキーを押してください。"
110 A$=""
120 A$=INKEY$
130 IF A$="" GOTO 120
140 PRINT " [ ";A$;" ] ";"のキーが押されました。"
150 '
160 PRINT:PRINT"もう一度やりますか? ( Y=yes  N=no )"
170 B$=""
180 B$=INKEY$
190 IF B$="" GOTO 180
200 IF B$="Y" GOTO 210 ELSE END
210 PRINT:PRINT"さあ、もう一度"
220 GOTO 100
```

参照　INPUT

画面にキーボードの絵を表示

```
100 '************************************
110 '         ﾓｸﾞﾗﾀﾀｷ ﾀｲﾌﾟ ｹﾞｰﾑ
120 '************************************
130 CONSOLE 0,25,1,1
140 WIDTH 80,25
150 CLS 3
160 DIM CH$(4)
170 NO=0:CT=0:LM=1:TL=1
180 *STUP 'ｼﾞｭﾝﾋﾞ
190 PRINT"    ┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐"
200 PRINT"    │1││2││3││4││5││6││7││8││9││0││-││^││¥│"
210 PRINT"    └─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘"
220 PRINT"     ┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐  "
230 PRINT"     │Q││W││E││R││T││Y││U││I││O││P││@││[│  "
240 PRINT"     └─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘  "
250 PRINT"      ┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐ "
260 PRINT"      │A││S││D││F││G││H││J││K││L││;││:││]│ "
270 PRINT"      └─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘ "
280 PRINT"       ┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐┌─┐   "
290 PRINT"       │Z││X││C││V││B││N││M││,││.││/││ │   "
300 PRINT"       └─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘└─┘   "
310 LOCATE 14,14:PRINT"****************************************"
320 LOCATE 14,15:PRINT"****                                ****"
330 LOCATE 14,16:PRINT"****  ﾓ ｸﾞ ﾗ ﾀ ﾀ ｷ  ﾀ ｲ ﾌﾟ  ｹﾞ ｰ ﾑ  ****"
340 LOCATE 14,17:PRINT"****                                ****"
350 LOCATE 14,18:PRINT"****************************************"
```

```
360 CH$(1)="1234567890-^¥"
370 CH$(2)="QWERTYUIOP@["
380 CH$(3)="ASDFGHJKL;:]"
390 CH$(4)="ZXCVBNM,./"
400 *SCORE 'スコア
410 IF HS=<SC THEN HS=SC
420 SC=CT
430 CT=0
440 LOCATE 14,13:PRINT"SCORE ";SC
450 LOCATE 29,13:PRINT"HI-SCORE ";HS
460 LOCATE 46,13:PRINT"COUNT ";CT;
470 BEEP:LOCATE 14,20:PRINT"ARE YOU READY?  HIT SPACE BAR      "
480 KY$=INKEY$:IF KY$="" THEN 480
490 BEEP:LOCATE 14,20:PRINT"                                    "
500 LOCATE 32,19:PRINT"┌───┐"
510 LOCATE 32,20:PRINT"│   │"
520 LOCATE 32,21:PRINT"└───┘"
530 *MAIN 'メイン
540   TH=VAL(MID$(TIME$,1,2))
550   TM=VAL(MID$(TIME$,4,2))
560   TS=VAL(MID$(TIME$,7,2))
570   TI=TH*3600+TM*60+TS
580   IF RF=0 THEN RANDOMIZE INT(TI/2)+1:RF=1
590   TI$=RIGHT$("0"+MID$(STR$(TH),2),2)+":"
600   TI$=TI$+RIGHT$("0"+MID$(STR$(TM+TL),2),2)+":"
610   TI$=TI$+RIGHT$("0"+MID$(STR$(TS),2),2)
620   ON TIME$=TI$ GOSUB *TLMT
630   TIME$ ON
640 *LOOP 'ループ
650   NO=INT(RND*47)+48
660   IF 60=<NO AND NO<=63 THEN NO=NO-16
670   QS$=CHR$(NO)
```

（360〜390）キーボードの配置を変数に代入

（410〜460）得点(CT)、前回スコア(SC)、ハイスコア(HS)の計算と表示

470〜490行はスペースが押されるまでループで待ち、押されると案内を消して続行する

（540〜580）問題を出すパターンが同じにならないように時計機能を使ってランダムにする

（590〜630）1分間のタイマーをセット

（650〜670）問題の文字をランダムに作り変数(QS$)に代入して表示

●ブラインド・タッチのマスターやタイプ練習ソフトにも

画面に表示されたキーを、実際のキーボードで1分間にどれだけ正確に打つことができるかという、モグラたたきゲームを応用したタイプ練習プログラム。

```
680    LOCATE 34,20:PRINT QS$
690    FOR I=1 TO 4
700      IF INSTR(CH$(I),QS$)<>0 THEN 710 ELSE 760
710        RV=INSTR(CH$(I),QS$)
720        X1=((RV-1)*5)+I+2:Y1=(I-1)*3
730        X2=((RV-1)*5)+I+6:Y2=(I-1)*3+2
740        COLOR@ (X1,Y1)-(X2,Y2),6
750      I=5
760    NEXT I
770    LOCATE 46,13:PRINT"COUNT ";CT;
780    TT=0
790    KY$=""
800    IF TT<1000 THEN GOTO 810 ELSE 880
810      KY$=INKEY$
820      TT=TT+1
830      IF KY$="" THEN 800
840    IF QS$=KY$ THEN 850 ELSE 880
850      CT=CT+1
860      COLOR@ (X1,Y1)-(X2,Y2),7
870      GOTO 900
880    BEEP
890      COLOR@ (X1,Y1)-(X2,Y2),7
900 GOTO *LOOP
910 *TLMT
920 BEEP
930 TIME$ OFF
940 LOCATE 14,20:PRINT"ﾓｳｲﾁﾄﾞ ﾁｬﾚﾝｼﾞ ? ( 'Y'es or 'N'o )";
950 KY$=INKEY$:IF KY$="" THEN 950
960 IF KY$<>"N" OR KY$<>"n" THEN CLS:GOTO *STUP
970 END
```

（690〜760）画面上のキーボードの文字の位置を計算して、黄色に色を変えて問題を出す

（780〜830）キーボードから答えの入力を待つ。一定時間経過すると次の問題に移ってしまう

（840〜900）答えが合っていれば得点(CT)を上げ、そうでなければそのまま、黄色の文字を白に戻して、次の問題に移る

（910〜970）620〜630行で１分経過を監視して、このルーチンへジャンプする。１分経過すると１回のゲームが終了

スペースバーを押してスタート。ディスプレイ画面上のキーボードが黄色になったキーを押す。キーボードを見ないで打つブラインド・タッチのマスターや、タイプ練習テキストに習って出題するように改良すれば、タイプ練習ソフトにも。

KEY／KEYLIST（キー／キーリスト）

■ f キーに文字列を定義／定義された内容を表示

書式　① KEY　キー番号，文字列
　　　　② KEYLIST

➔書式①は、キーボードの上部にある10個のファンクションキー（f キー）のうち、「キー番号」で指定したファンクションキーに、指定した「文字列」を定義する。以降、ファンクションキーを押すと、定義された文字列がキーボードから入力されたのと同様の働きをする。

➔「文字列」は、最大15文字までの文字およびコントロール文字（画面制御などに用いられる文字で、キーボード上には配置されていない）を定義することができる。

➔コントロール文字はCHR$関数を使って定義する。

➔「文字列」には日本語文字列は含まれない。つまり、ファンクションキーに日本語文字列を定義することはできない。

➔書式②は、ファンクションキーに定義された内容をすべて画面に表示させる。

用例

```
100 KEY 1,"AUTO 100,10"
110 KEY 2,"LOAD "+CHR$(&H22)+"B:"
120 KEY 3,"SAVE "+CHR$(&H22)+"B:"
130 KEY 4,"LIST "
140 KEY 5,"LIST .-"+CHR$(13)
150 KEY 6,"DELETE "
160 KEY 7,"RENUM 100,100"+CHR$(13)
170 KEY 8,"EDIT "
180 KEY 9,"EDIT ."+CHR$(13)
190 KEY 10,"CONT"+CHR$(13)
```

参照　CHR$

■漢字ファンクションキーをシミュレートする

```
100 '************************************
110 ' 漢字ファンクションキー
120 '************************************
130 CONSOLE 0,25,0,0
140 WIDTH 80,25
150 CLS 3
160 DIM FK$(10)
170 *STUP 'ｼﾞｭﾝﾋﾞ ----------------------
180 RESTORE *FDAT
190 FOR I=1 TO 10
200   READ FK$(I)
210   PRINT FK$(I)
220 NEXT I
230 FOR I=1 TO 10
240   KEY I,"=^|"+RIGHT$(STR$(I),1)+CHR$(13)
250 NEXT I
260 *MAIN 'ﾒｲﾝ ---------------------------
270 GOSUB *FKDP
280 GOSUB *FKIN
290 LOCATE 10,15:PRINT"You select No.";NO;"Key."
300 LOCATE 10,20:INPUT "Do you retry? ( 'Y'es or
'N'o )";ANS$
310 IF ANS$="Y" OR ANS$="y" THEN GOTO *MAIN
320 GOSUB *REST
330 CONSOLE 0,25,1,1
340 END
350 *FKDP 'ﾌｧﾝｸｼｮﾝｷｰ ﾋｮｳｼﾞ ---------------
360 FOR I=1 TO 5
370   LOCATE (I-1)*7+4,24:PRINT FK$(I);
380   COLOR@ ((I-1)*7+4,24)-((I-1)*7+9,24),4
390 NEXT I
400 FOR I=6 TO 10
410   LOCATE (I-1)*7+7,24:PRINT FK$(I);
420   COLOR@ ((I-1)*7+7,24)-((I-1)*7+12,24),4
430 NEXT I
440 RETURN
450 *FKIN 'ﾌｧﾝｸｼｮﾝｷｰ ﾆｭｳﾘｮｸ --------------
460 LOCATE 10,10:PRINT "ファンクションキーで選択
してください";
```

(180〜220) 680行の漢字データを変数(FK$)に代入

(230〜250) ファンクションキーに実際に設定される隠しコマンド

(240) 隠しコマンド文字列 / ファンクションキーの番号

(360〜430) f.1〜f.5、f.6〜f.10までの漢字ファンクションを表示

```
470 KY$=""
480 WHILE KY$=""
490   INPUT "",KY$
500   IF LEFT$(KY$,3)<>"=^¦" THEN KY$=""
510 WEND
520 LOCATE 46,10:PRINT "     "
530 NO=VAL(RIGHT$(KY$,1))
540 IF NO=0 THEN NO=10
550 RETURN
560 *REST 'ﾌｧﾝｸｼｮﾝｷｰ ﾘｾｯﾄ ---------------
570 KEY 1,"load "+CHR$(34)
580 KEY 2,"auto "
590 KEY 3,"go to "
600 KEY 4,"list "
610 KEY 5,"run "+CHR$(13)
620 KEY 6,"save "+CHR$(34)
630 KEY 7,"key "
640 KEY 8,"print "
650 KEY 9,"files "
660 KEY 10,"cont "+CHR$(13)
670 RETURN
680 *FDAT 'ﾌｧﾝｸｼｮﾝｷｰ ﾃﾞｰﾀ ---------------
690 DATA 一番目,二番地,三人様,四軒分,五丁目
700 DATA 六で無,七五三,八百屋,九官鳥,十人力
```

ファンクションキーの入力とチェック（470〜510行）

隠しコマンドでチェックする（500行）

プログラムが終了したら元のファンクションキーに戻す（570〜660行）

漢字ファンクションに設定するデータ。自由に設定できる（690〜700行）

●キーは隠しコマンド文字列

BASICでは、ファンクションキーに漢字を設定することはできないが、このプログラムで簡単に「漢字ファンクションキー」をシミュレートできる。

タネは「隠しコマンド文字列」である。240行の「=^¦」は、キーボードの右上のシフト位置にある3文字。ふだん使われない文字の組み合わせとファンクションキー番号に対応する数字を、KEY文でほんもののファンクションキーに設定している。ファンクションキーを押すと、これらの文字列が入力されるので、その文字列によって判断している。

290行にファンクションキーに対応する作業をプログラムすれば、「漢字ファンクションキー」が実用的に使える。

3．実行制御

LET（レット）

■左辺の変数に右辺の式の値を代入する

書式　① LET　変数＝式

　　　　② 変数＝式

➔ LET は、左辺（＝の左側）の変数に、右辺（＝の右側）の式の値や変数の内容を代入するときに使う。

➔ LET は省略することができ、一般的には、書式②のように使う。

➔ 式の内容は数値、文字列どちらの型でもよいが、左辺の変数と同じ型でなくてはならない。文字型から数値型、またその逆への代入はできない。精度の異なる数値変数へ代入する場合は、右辺の式の値を左辺の変数の精度に合わせて代入されるので注意する。

用例

```
100 LET A=100
110 LET B=100*2*(32/4)
120 LET C=SQR(2)
130 LET D=A+B
140 LET E$="ABCD*-*/%¥ アイウエオ"
150 LET F$="日本語文字"
160 LET G$=E$+F$
[redacted] D
[redacted] G$
[redacted]
200 I=100*2*(32/4)
[redacted](2)
[redacted]
230 E$="ABCD*-*/%¥ アイウエオ"
```

```
240 M$="日本語文字"
250 N$=E$+F$
260 PRINT H,I,J,K
270 PRINT L$,M$,N$
```

REM（リマーク）

■プログラム中に注釈文を入れる

書式 ① REM　コメント文

② '　コメント文

➔ REM 文は非実行文といい、プログラムを実行する際に読み飛ばされて、実行に影響を与えない。

➔「コメント文」には任意の文字列が書ける。そのため、プログラム中に必要な注意や見出しを注釈行として入れておくことができる。

➔ REM 文は省略型として、書式②のように「'（アポストロフィー）」を使うことができる。一般的にはこの省略型を使って、コメントや飾りを置いてアクセントとすることが多い。

➔ REM 文の後ろは、「：（コロン）」で区切って他の文を続けるマルチステートメントにすることはできない。続けても実行されない注釈文の一部としてしまうので注意する。

用例

```
100 REM --------------- PROGRAM START ---------------
110 REM ABCDEFG ¥!"#$%& ｱｲｳｴｵｶｷｸ 123456 +-*/=
120 REM 注釈行
130 REM :PRINT"PROGRAM STOP"
140 ' --------------- PROGRAM START ---------------
150 ' ABCDEFG ¥!"#$%& ｱｲｳｴｵｶｷｸ 123456 +-*/=
160 ' 注釈行
170 ' :PRINT"PROGRAM STOP"
```

STOP (ストップ)

■プログラムの実行を一時中断する

書式　① STOP
　　　② STOP キー

→STOP 文はプログラムの実行を一時中断する。

→プログラム中のどこに置いても、STOP 文を実行すると、

　　Break in　行番号

と表示し、実行をストップした「行番号」を示してコマンドレベルに戻る。

→書式②のように、実行中のプログラムを STOP (ストップ)キーを押すことによって一時中断することもできる。

→一時中断してダイレクトモードで変数の内容などを確認した後、CONT コマンドでプログラムの実行を再開することができる。

→INPUT 文などの入力命令の実行中にストップした場合は、実行を再開することはできない。

→END 文と異なり、STOP 文はオープンされているファイルを閉じないので、実行を停止してファイルを操作する場合などには注意する。

用例

```
100 N=0
110 FOR I=1 TO 10000
120   N=N+1
130   IF N/100=INT(N/100) THEN PRINT "N=";N" デ゛ス":STOP
140 NEXT I
```

参照　CONT／END

END（エンド）

■プログラムの終点を示す

書式　END

➔プログラムの終点を示す。END文を実行するとプログラムの実行を停止し、オープンされているすべてのファイルをクローズしてコマンドレベルに戻る。

➔END文は1つのプログラム中にいくつ置いてもよく、またどこに置いてもよい。

➔プログラムの最後（最大の行番号の付けられた行）のEND文は省略することができるが、この場合ファイルのクローズは行なわれないので注意する。

➔END文のある行より後の行にプログラムを続けて書くこともできるので、本体プログラムとサブルーチン群とを分けてプログラムを書くこともできる。

➔END文は比較的自由に使うことができるが、1つのプログラムの本体の最後に1つだけ使うほうが見やすいプログラムになる。

用例

```
100 '=== MAIN ROUTINE ===
110 FOR I=1 TO 100
120   GOSUB *SUB
130   PRINT N;
140   IF N/10=INT(N/10) THEN PRINT
150 NEXT I
160 END
170 '--- SUB ROUTINE ---
180 *SUB
190 N=N+1
200 RETURN
```

参照　STOP

GOTO／GO TO（ゴーツー）

■指定された行へプログラムの実行を移す

書式　① GOTO　行番号

② GOTO　＊ラベル名

➔書式①は、「行番号」で指定された行へプログラムの実行を移す（ジャンプするともいう）。

➔書式②は、「＊ラベル名」で指定されたラベルのある行へプログラムの実行を移す（ジャンプする）。

➔「GOTO」は「GO TO」と書いてもいいが、このとき「GO」と「TO」の間のスペースは1文字分だけであること。2文字分以上スペースをあけるとGOTO文とはみなされず、エラーになるので注意する。

用例

```
100 N=0
110 M=100
120 *SUMPLE.1
130 N=N+1
140 PRINT N;
150 IF N/10=INT(N/10) THEN PRINT
160 IF N=100 THEN GOTO 190
170 GOTO *SUMPLE.1
180 '-------------------
190 *SUMPLE.2
200 M=M+1
210 PRINT M;
220 IF M/10=INT(M/10) THEN PRINT
230 IF M=200 THEN GOTO 250
240 GOTO *SUMPLE.2
250 END
```

参照　GOSUB～RETURN

GOSUB～RETURN（ゴーサブ～リターン）

■指定したサブルーチンを呼び出して実行する

書式
① GOSUB　行番号
② GOSUB　＊ラベル名
　　～
③ RETURN
④ RETURN　行番号
⑤ RETURN　＊ラベル名

➔サブルーチンとは、プログラムの本体であるメインルーチンから独立したプログラムで、RETURN 文で終わっているものをいう。

➔サブルーチンは、書式①のように「行番号」を指定するか、書式②のように「＊ラベル名」を指定して、GOSUB 文で呼び出す。

➔サブルーチンからは RETURN 文でメインルーチンにプログラムの実行を戻す。

➔サブルーチンには RETURN 文を複数置くことができるが、最初に出会った RETURN 文によってメインルーチンへ戻される。

　また、GOSUB による呼び出しのない RETURN 文に出会う（メインルーチンに RETURN 文が存在する場合）と、RETURN without GOSUB エラーとなるので注意する。

➔ RETURN 文では、書式③のようになにも指定しないとサブルーチンを呼び出した GOSUB 文の次の命令に実行を戻す。

➔書式④、書式⑤のように「行番号」、または「＊ラベル名」を指定して、特定の行に強制的に実行を戻すこともできる。

用例

```
100 'MAIN ---------------
110 GOSUB 200
```

```
120   PRINT "ﾓﾄﾞﾘﾏｼﾀ":PRINT
130 GOSUB *SUBROUTINE.1
140   PRINT "ﾓﾄﾞﾘﾏｼﾀ":PRINT
150 GOSUB *SUBROUTINE.2
160   PRINT "160 ｷﾞｮｳ ﾆﾊ ﾓﾄﾞﾘﾏｾﾝ"
170   PRINT "170 ｷﾞｮｳ ﾆ ﾓﾄﾞﾘﾏｼﾀ"
180 END
190 'SUBROUTINE ---------
200   PRINT "200 ｷﾞｮｳ ﾆ ｷﾏｼﾀ"
210   RETURN
220 *SUBROUTINE.1
230   PRINT "SUBROUTINE.1 ﾆ ｷﾏｼﾀ"
240   RETURN
250 *SUBROUTINE.2
260   PRINT "SUBROUTINE.2 ﾆ ｷﾏｼﾀ"
270   RETURN 170
```

参照　GOTO

ON GOTO／ON GOSUB
（オン　ゴーツー／オン　ゴーサブ）

■番号で指定された行に分岐する

書式
① ON　番号　GOTO　行番号，行番号，………
② ON　番号　GOTO　＊ラベル名，＊ラベル名，……
③ ON　番号　GOSUB　行番号，行番号，………
④ ON　番号　GOSUB　＊ラベル名，＊ラベル名，
………

➔「番号」は1から始まる整数で、数値変数でも式でも指定できる。数値変数はその内容、式は計算した結果の値で分岐先を指定する。

➔分岐先の指定は、書式①③のように「行番号」、また書式②④のように「＊ラベル名」で指定することができる。

➔分岐先の「行番号」「＊ラベル名」は、GOTO または GOSUB に

続けて、1から順番に対応して「，(カンマ)」で区切って並べる。たとえば、「番号」が2なら、2番目に指定された分岐先にプログラムの実行を移す。

➔「番号」の値が負になった場合は Illegal function call エラーとなる。しかし、0または分岐先の行番号、ラベル名の並びの数より大きくなってもエラーとはならず、次の行へプログラムの実行が移るだけなので、注意する。

用 例

```
100 'MAIN ---------------
110 INPUT "1 カラ 5 ノ スウチ ヲ オシテ クダサイ";N
120 ON N GOTO 210,220,230,240,250
130 PRINT "1 カラ 5 ノ スウチ デハ アリマセンデシタ"
140 PRINT "ツギニ イキマス"
150 PRINT
160 INPUT "6 カラ 10 ノ スウチ ヲ オシテ クダサイ";N
170 ON N-5 GOSUB 260,270,280,290,300
180 END
190 '--------------------
200 '
210   PRINT "210 ギョウ ニ キマシタ":GOTO 140
220   PRINT "220 ギョウ ニ キマシタ":GOTO 140
230   PRINT "230 ギョウ ニ キマシタ":GOTO 140
240   PRINT "240 ギョウ ニ キマシタ":GOTO 140
250   PRINT "250 ギョウ ニ キマシタ":GOTO 140
260   PRINT "260 ギョウ ニ キマシタ":RETURN
270   PRINT "270 ギョウ ニ キマシタ":RETURN
280   PRINT "280 ギョウ ニ キマシタ":RETURN
290   PRINT "290 ギョウ ニ キマシタ":RETURN
300   PRINT "300 ギョウ ニ キマシタ":RETURN
```

参 照　GOTO／GOSUB～RETURN

IF～THEN～ELSE(イフ～ゼン～エルス)

IF～GOTO～ELSE(イフ～ゴーツー～エルス)

■条件によってプログラムの実行を制御する

書式　① IF　条件式　THEN　文または行番号
　ELSE　文または行番号
② IF　条件式　GOTO　行番号
　ELSE　文または行番号

➔「条件式」が正しい場合(真＝0以外の値)、THEN または GOTO 文以降を実行し、正しくない場合(偽＝0の値)は ELSE 文以降に実行が移される。

➔ THEN 文、ELSE 文の後には、「文」または「行番号」を置いてプログラムを実行することができる。

➔「行番号」の代わりに「＊ラベル名」を指定しても、プログラムの実行を移すことができる。

➔条件式は、一般に論理式で真偽や大小を判断させるが、ふつうの算術演算の結果の0、または0以外の値によって実行を制御することもできる。

用例

```
100 INPUT "1 ｶﾗ 10 ﾉ ｽｳﾁ ｦ ｵｼﾃ ｸﾀﾞｻｲ";N
110 IF N<5 THEN PRINT "ｿﾚ ﾊ 5 ﾖﾘ ﾁｲｻｲ ﾃﾞｽ"
ELSE PRINT"ｿﾚ ﾊ 5 ｲｼﾞｮｳ ﾃﾞｽ"
120 '
130 PRINT
140 INPUT "1 ｶﾗ 10 ﾉ ｽｳﾁ ｦ ｵｼﾃ ｸﾀﾞｻｲ";N
150 IF N<5 THEN 220 ELSE 230
160 '
170 PRINT
```

```
180 INPUT "1 カラ 10 ノ スウチ ヲ オシテ クダサイ";N
190 IF N-5 GOTO 250 ELSE 260
200 END
210 '--------------------
220 PRINT "ソレ ハ 5 ヨリ チイサイ デス":GOTO 160
230 PRINT"ソレ ハ 5 イジョウ デス":GOTO 160
240 '
250 PRINT "ソレ ハ 5 デハ アリマセン":GOTO 200
260 PRINT "ソレ ハ 5 デス":GOTO 200
```

参照 GOTO

FOR—TO—STEP～NEXT
（フォー　ツー　ステップ～ネクスト）

■一連の命令を指定された回数だけ繰り返す

書式　FOR　数値変数＝初期値　TO　終了値
　　　　　　　　　　　　　　　STEP　増分

　　　　　～

　　NEXT　数値変数

➔ FOR～NEXT 文は、次のような順序で実行される。

1. 「数値変数」はループカウンタ（繰り返す回数を計算するための変数）として使われ、最初に「初期値」に設定される。
2. FOR 文と NEXT 文に囲まれた範囲のプログラムが実行される。
3. ループカウンタを「増分」だけ増減して、「終了値」と比較して、ループカウンタの値が「終了値」に達していなければ、2に戻り同じ処理が繰り返される。

➔ STEP 文が省略された場合、「増文」は＋1とみなされ、1ずつ増えていく。また、「増分」には負の値も指定でき、この場合は指定され

た数だけループカウンタを減じていく。

➔ FOR～NEXT 文は、「入れ子」構造にすることもできる。つまり、1つの FOR～NEXT 文の中に、もう1つの FOR～NEXT 文を置いて、多重に繰り返しを行なうようにすることができる。

この場合は、ループカウンタとしての「数値変数」は、別のものにしなければならず、また一方の FOR～NEXT 文は、必ず他方の FOR～NEXT 文の内部に含まれていなければならないので、注意する。

用例

```
100 PRINT "1 ツ゛ツ カソ゛エマス"
110 FOR I=0 TO 99
120   PRINT I;
130 NEXT I
140 PRINT:PRINT
150 PRINT "1 ツ゛ツ キ゛ャク ニ カソ゛エマス"
160 FOR I=99 TO 0 STEP -1
170   PRINT I;
180 NEXT I
190 PRINT:PRINT
200 INPUT "STEP ハ イクツニ シマスカ (1 カラ 10)";N
210 FOR I=0 TO 99 STEP N
220   PRINT I;
230 NEXT I
240 PRINT:PRINT
250 INPUT "イレコ ニ シマス  ナニカ キー ヲ オシテクダ゛サイ :";K$
260 FOR I=0 TO 9
270   FOR J=0 TO 9
280     PRINT I;"-";J;
290   NEXT J
300   PRINT
310 NEXT I
320 END
```

WHILE～WEND（ホワイル～エンド）

■条件が成立している間、一連の命令を繰り返す

書式　WHILE　条件式
　　　　　～
　　　WEND

➔「条件式」が正しい（真＝0以外の値）場合、WHILE～WENDに囲まれた範囲の命令を実行し、WEND文からWHILE文へ戻って再度条件を判断する。これを繰り返すことによって1つのループとする。

➔この繰り返しは、「条件式」が成立している間続けられ、正しくなくなる（偽＝0の値）と、WEND文の次の文に実行が移される。

➔ WHILE文の後ろの「条件式」で真偽を判断するので、最初に「条件式」が偽と判定されると、WEND文までの命令は一度も実行されないので、注意する。

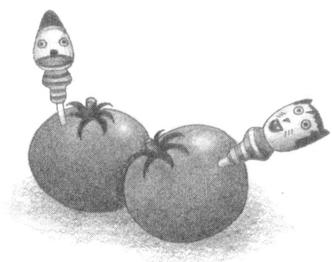
➔ WHILE～WEND文は、「入れ子」構造にすることもできる。つまり、1つのWHILE～WEND文の中にもう1つのWHILE～WEND文を置いて、多重に繰り返しを行なうようにすることができる。一方のWHILE～WEND文は、必ず他方のWHILE～WEND文の内部に含まれていなければならない。

用例

```
100 PRINT "1 ヅツ カゾエマス"
110 N=0
120 WHILE N<100
130   PRINT N;
140   N=N+1
150 WEND
160 PRINT:PRINT
170 N=100
```

```
180 PRINT "1 ﾏﾃﾞ ｷﾞｬｸ ﾆ ｶｿﾞｴﾏｽ"
190 WHILE N
200   PRINT N;
210   N=N-1
220 WEND
230 PRINT:PRINT
240 INPUT "ｲﾚｺ ﾆ ｼﾏｽ  ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ :";K$
250 I=0
260 WHILE I=<10
270   J=0
280   WHILE J=<10
290     PRINT I;"-";J;
300     J=J+1
310   WEND
320   I=I+1
330   PRINT
340 WEND
350 END
```

参照　FOR～NEXT



4．変数定義

DEFINT（デファインインテジャー）

DEFSNG（デファインシングル）

DEFDBL（デファインダブル）

DEFSTR（デファインストリング）

■変数の型を定義する

書式　① DEFINT　変数名の範囲
　　　② DEFSNG　変数名の範囲
　　　③ DEFDBL　変数名の範囲
　　　④ DEFSTR　変数名の範囲

➔「変数名の範囲」で指定されたアルファベット文字で始まる変数名の変数を、各型の変数に定義する。定義によって、各変数は型宣言文字（%、!、#、$）を付加しなくてもよくなる。

➔ DEFINT は整数型に、DEFSNG は単精度実数型に、DEFDBL は倍精度実数型に、DEFSTR は文字型に、それぞれ宣言する。

➔「変数名の範囲」で指定される文字は英字1文字で、一度に複数個連続して指定できる。複数個を指定する場合は、始まりの英文字と終わりの英文字の間を「－（マイナス）」でつなぐ。また、連続しない場合は、「，（カンマ）」で区切って並べて指定することもできる。

➔型が定義されていない変数は、単精度実数型とみなされる。また、

型宣言文字を付加した場合は、そちらが優先される。

用例

```
100 DEFINT A
110 DEFSNG B-C
120 DEFDBL D
130 DEFSTR E,J
140 '--------------------
150 A=.4                :PRINT A
160 A=.5                :PRINT A
170 B=123456!           :PRINT B
180 C=123456789#        :PRINT C
190 D=12345678901234#   :PRINT D
200 E="ABCDEFGHIGK"     :PRINT E
210 F=123456!           :PRINT F
220 J="日本語文字"       :PRINT J
230 Z=D                 :PRINT Z
```

参照 24ページBASICの文法「⑨定数、⑩変数」

DEF FN（デファイン ファンクション）

■ユーザー定義関数を定義する

書式 DEF FN関数名 引数＝関数定義式

➔「FN関数名」で名づけられたユーザー定義関数を、「関数定義式」で設定された演算を行なう関数として定義する。

➔「関数名」は、変数と同じ条件で名づけ、使用時にはFNと合わせて1つの関数名として扱われる。

➔「引数」は、「関数定義式」の中だけで使われるもので、プログラム中の他の実際の変数には影響しないので、適当な名まえを使うことができる。

➔「関数定義式」は、「引数」を使った実際の演算の式を書くが、「引数」以外の変数名を演算する場合は、ユーザー定義関数を使用した

ときの変数の値を使って演算される。

➔使用する際は、FN も含めた関数名で使い (FNAAA (10) などとする)、引数として実際のデータを指定すれば、データに対して演算を行ない、結果を返す。

➔ DEF　FN 文は非実行文であるが、プログラム中でユーザー定義関数を使用する前に定義しておかなければならない。

用 例

```
100 DEF FNA(X)=X*2
110 DEF FNB(Y)=Y*Y
120 DEF FNC$(X$)=X$+" サマ"
130 DEF FND(Y$)=LEN(Y$)*10
140 '--------------------
150 INPUT "スウチ ヲ イレテクダサイ";N
160 PRINT "FNA(X) =";FNA(N)
170 PRINT "FNB(Y) =";FNB(N)
180 INPUT "モジ ヲ イレテクダサイ";A$
190 PRINT "FNC$(X$) =";FNC$(A$)
200 PRINT "FNC(Y$) =";FND(A$)
210 END
```

DIM（ディメンション）

■配列の変数名と大きさを宣言する

書式　①　DIM　変数名（添え字の最大値）
②　DIM　変数名（添え字の最大値，添え字の最大値，··········）

➔大量のデータを効率よく扱うために、「変数名」に「添え字」で指定される番号を付けて配列を宣言する。「変数名」は、変数名の付け方の条件に合っていれば、自由に付けることができる。

➔「添え字」の最大数は、使う配列の要素の数を指定するが、1個の配列に対して指定できる要素は、整数型は32767個、単精度実数型は16383個、倍精度実数型は8191個までとなっている。

➔多次元配列といわれる配列を使用するときに、書式②のようにいくつかの「添え字の最大値」を「，（カンマ）」で区切って指定する。

➔添え字の最小値は0であるが、OPTION BASE 文で、1または0に指定することができる。

➔DIM 文が実行されると、すべての配列の要素は、数値の場合0、文字の場合はヌルストリング（空＝カラの文字列）に設定される。

➔配列変数は DIM 文で宣言されなくても使うことができるが、この場合、添え字の最大値は10とみなされる。

用例

```
100 DIM A(100)
110 DIM B(10,10)
120 DIM C$(100)
130 DIM D(10),E$(10),F$(10)
140 '--------------------
150 FOR I=1 TO 100
160   A(I)=I*2
```

```
170 NEXT I
180 FOR I=1 TO 10
190   FOR J=1 TO 10
200     B(I,J)=I*J
210   NEXT J
220 NEXT I
230 N=32
240 FOR I=1 TO 100
250   N=N+1:IF 126<N THEN N=32
260   C$(I)=CHR$(N)
270 NEXT I
280 FOR I=1 TO 10
290   READ NO,NA$,TL$
300   D(I)=NO
310   E$(I)=NA$
320   F$(I)=TL$
330 NEXT I
340 '--------------------
350 FOR I=1 TO 100
360   PRINT A(I);
370 NEXT I
380 PRINT
390 FOR I=1 TO 10
400   FOR J=1 TO 10
410     PRINT B(I,J);
420   NEXT J
430   PRINT
440 NEXT I
450 PRINT
460 FOR I=1 TO 100
470   PRINT C$(I);
480 NEXT I
490 PRINT:PRINT:PRINT "--------- TELEPHONE LIST"
500 FOR I=1 TO 10
510   PRINT Γ(I),E$(I),F$(I)
520 NEXT I
530 '--------------------
540 DATA 1,AKIKO,033-123-4567,2,ERIKO,098-000-1111
550 DATA 3,KEIKO,03-100-1000,4,SIZUKO,045-111-2222
560 DATA 5,TOSIKO,012-123-1234,6,NANAE,03-222-3333
570 DATA 7,HIROMI,06-456-7890,8,MARIKO,05-333-4444
580 DATA 9,MIEKO,099-98-7654,10,YUKI,0123-444-5555
```

参 照　OPTION BASE／ERASE

OPTION BASE（オプション ベース）

配列の添え字の下限を宣言する

書式　OPTION BASE 下限の値

➔配列変数で使う要素の番号である添え字の下限を、0または1に宣言する。

➔通常添え字の下限は0であるが、OPTION BASE文で1に設定することもできる。「下限の値」は、0または1でなければならない。

➔この宣言は、プログラム中で配列宣言が行なわれる前、または配列変数が用いられる前に実行されていなければならない。

➔一度宣言すると、再度宣言することはできない。

➔ OPTION BASE文で宣言された「下限の値」は、CLEARまたはNEWコマンドを実行すると解除され、0になるが、それまで変更はできない。

用例

```
100 OPTION BASE 1
110 DIM A(100)
120 '--------------------
130 FOR I=1 TO 100
140   A(I)=SQR(I)
150 NEXT I
160 '--------------------
170 FOR I=1 TO 100
180   PRINT A(I),
190 NEXT I
```

参照　DIM／CLEAR／NEW

ERASE（イレーズ）

配列を消去する

書式　ERASE　配列変数名

➔「配列変数名」で指定した配列を消去する。

➔配列を消去すると、消去した配列に割り当てられていたメモリエリアが解放されるので、フリーエリアが広くなり、他の目的に使用することができるようになる。

➔配列を消去すると、消去した配列名で再度配列を宣言することができるので、配列の要素すべての内容をクリアすることができる。

用例

```
100 DIM A(10)
110 '--------------------
120 FOR I=1 TO 10
130   A(I)=SQR(I)
140 NEXT I
150 FOR I=1 TO 10
160   PRINT A(I),
170 NEXT I
180 PRINT:PRINT
190 '--------------------
200 ERASE A
210 DIM A(10)
220 FOR I=1 TO 10
230   PRINT A(I),
240 NEXT I
250 '--------------------
260 FOR I=1 TO 10
270   A(I)=SIN(I)
280 NEXT I
290 PRINT:PRINT
300 FOR I=1 TO 10
310   PRINT A(I),
320 NEXT I
```

参照　DIM

SWAP（スワップ）

■2つの変数の値を交換する

書式　SWAP　変数，変数

➔指定された「変数」の内容を交換する。

➔両方の「変数」の型は必ず同じでなければならない。違う場合は、Type mismatch エラーが起こる。

➔変数の型が一致していれば、配列変数の内容も交換することができる。

用例

```
100 A=0
110 B=1
120 PRINT "A =";A,"B =";B
130 SWAP A,B
140 PRINT "A =";A,"B =";B
150 PRINT:PRINT
160 '--------------------
170 A$="AAA"
180 B$="BBB"
190 PRINT "A$ =";A$,"B$ =";B$
200 SWAP A$,B$
210 PRINT "A$ =";A$,"B$ =";B$
```

● SWAP ―――― 応用プログラム

隣り合ったデータを次つぎと比較して並べ替える

```
100 '**************************************
110 '              バブルソート
120 '**************************************
130 CONSOLE 0,25,1,0
140 WIDTH 80,25
```

```
150 CLS
160 DIM DT(20)
170 FOR I=1 TO 20
180   READ DT(I)
190 NEXT I
200 '------------------------------------
210                *MAIN
220 '------------------------------------
230 CLS
240 FOR I=1 TO 10
250   PRINT "No.";I;DT(I);"     No.";I+10;DT(I+10)
260 NEXT I
270 PRINT:PRINT"ｺﾉ ﾃﾞｰﾀ ｦ ｼｮｳｼﾞｭﾝﾆ ﾅﾗﾋﾞ ｶｴﾙﾄ"
280 'ﾊﾞﾌﾞﾙ ｿｰﾄ -------------------------
290 FOR I=20 TO 2 STEP -1
300   EF=0
310   FOR J=1 TO I-1
320     IF DT(J)>DT(J+1) THEN SWAP DT(J),DT(J+1):EF=1
330   NEXT J
340   IF EF=0 THEN I=2
350 NEXT I
360 '------------------------------------
370 FOR I=1 TO 10
380   PRINT "No.";I;DT(I);"     No.";I+10;DT(I+10)
390 NEXT I
400 END
410 DATA 15,36,21,89,75,14,32,90,46,81
420 DATA 55,73,82,16,40,64,82,19,28,68
```

配列宣言
データをセットする

並べ替え前のデータの表示

並べ替えを行なう

並べ替えた結果の表示

●大小が逆になっていたらSWAPでデータを入れ替える

もっともわかりやすいソート（並べ替え）法である「バブルソート」のプログラムである。290行から350行でソートを行なっている。隣り合ったデータを次つぎと比較していき、大小が逆になっていたらSWAP文を使ってデータを入れ替えていく。320行で、この比較とデータの入れ替えを行ない、410行、420行の20個のデータをソートしている。

データの数を増やしたり、配列変数を増やすと、より多くのデータをソートできる。また、他のプログラムに組み込んで応用することもできる。

RANDOMIZE（ランダマイズ）

■新しい乱数系列を設定する

書式　① RANDOMIZE　初期値
　　　　② RANDOMIZE

➔乱数の系列に新しい「初期値」を与えることによって、乱数系列を変更する。「初期値」は、−32768から32767までの数値で与える。変数や式として与えることもできる。

➔書式②では、「Random number seed(−32768 to 32767)?」とメッセージを表示して、「初期値」(seed：シーズ＝種）を求めてくるので、上の範囲内の値をキーボードから入力する。

➔同じ「初期値」を与えると、同じ乱数系列を発生する。

用例

```
100 RANDOMIZE
110 FOR I=1 TO 10
120   PRINT RND(I),
130 NEXT I
140 PRINT:PRINT
150 '---------------------
160 RANDOMIZE 0
170 FOR I=1 TO 10
180   PRINT RND(I),
190 NEXT I
200 PRINT:PRINT
210 '---------------------
220   TM=VAL(MID$(TIME$,4,2))
230   TS=VAL(MID$(TIME$,7,2))
240   TI=TM*60+TS
250 RANDOMIZE TI
260 FOR I=1 TO 10
270   PRINT RND(I),
280 NEXT I
```

参照　RND

CLEAR（クリア）

■変数を初期化しメモリレイアウトを設定する

書式　① CLEAR

② CLEAR　ダミーパラメータ, メモリの上限, スタックの大きさ, 配列のメモリエリアの大きさ

➔書式①は、すべての変数の初期化を行ない、変数の内容は数値変数は0に、また文字変数はヌルストリング（空＝カラの文字列）にする。

➔書式②は、BASICで使用するメモリレイアウトを設定するのに用いる。

➔「ダミーパラメータ」は、指定してもなんの効果もないので、一般に省略する。

➔「メモリの上限」は、BASICが使用するメモリの上限をセグメントアドレスで指定する。これによって、その番地以降に置かれたデータや機械語プログラムがBASICによって破壊されることがなくなる。

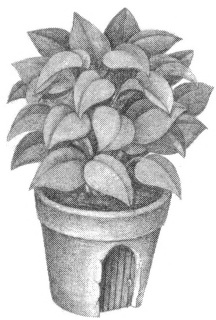
➔「スタックの大きさ」は、FOR～NEXT文、GOSUB文などに使用するスタック領域の大きさを、バイト数で指定する。リセット時の初期値は512バイトが設定されている。

➔「配列のメモリエリアの大きさ」は、数値配列のデータが格納されるメモリエリアの大きさを、バイト数で指定する。これによって、数値配列のデータエリアを縮小して、シンボルテーブルやストリング格納エリアのサイズを拡大することができる。

用　例

```
100 A=100:B$="ABC"
110 PRINT A,B$
120 CLEAR
130 PRINT A,B$
140 PRINT "フリーエリア =";FRE(1)
150 CLEAR ,&H1B00
160 PRINT "フリーエリア =";FRE(1)
170 PRINT "------------------------"
180 PRINT "ヘンスウ+フリーエリア =";FRE(2)
190 PRINT "     ハイレツ エリア =";FRE(3)
200 CLEAR ,,&H256,&H3840
210 PRINT "ヘンスウ+フリーエリア =";FRE(2)
220 PRINT "     ハイレツ エリア =";FRE(3)
```

参　照　FRE



5. テキスト画面制御

CONSOLE（コンソール）

スクロール範囲や画面モードを設定する

書式　CONSOLE　スクロール開始行，スクロール行数，ファンクションキー表示スイッチ，カラー／白黒スイッチ

➔「スクロール開始行」によって、何行目からスクロールを開始するか、「スクロール行数」によって何行分スクロールするかを指定して、テキスト画面のスクロール範囲（スクロールウィンドウ）を設定する。

➔「ファンクションキー表示スイッチ」は、1のとき、画面最下行に、ファンクションキーに登録されている文字列を表示する。0ではこれを表示しない。BASIC 起動直後は表示されるモード（1の指定と同じ）になっている。

➔「カラー／白黒スイッチ」は、1でテキスト画面をカラー表示のモードにし、0で白黒表示のモードに指定する。BASIC 起動直後は白黒表示モードになっている。

➔それぞれの指定は、指定する必要のないものは省略することができるが、それぞれの順番を崩すことはできないので、省略した部分にはなにも指定せずに、「,（カンマ）」のみを置く。

用例

```
100 CONSOLE 0,25,0,0
110 PRINT "ナニカ キー ヲ オシテクダサイ"
120 KY$=INKEY$:IF KY$="" THEN 120
```

```
130 CLS
140 '---------------------
150 CONSOLE 0,25,1,0
160 PRINT "ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
170 KY$=INKEY$:IF KY$="" THEN 170
180 CLS
190 '---------------------
200 CONSOLE 0,24,1,1
210 COLOR 4
220 FOR I=1 TO 25:PRINT STRING$(79,"#"):NEXT I
230 PRINT "ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
240 KY$=INKEY$:IF KY$="" THEN 240
250 CONSOLE 0,12,1,1
260 CLS
270 COLOR 7
280 PRINT "ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
290 KY$=INKEY$:IF KY$="" THEN 290
300 CONSOLE 0,25,1,1
310 CLS
```

WIDTH／WIDTH LPRINT
（ウィズス／ウィズス　エルプリント）

■文字の表示、出力の桁数や行数を設定する

書式　①　WIDTH　桁数，行数

②　WIDTH　"COM１：",　サイズ

③　WIDTH LPRINT　文字数

➔書式①は、テキスト画面に表示する文字の桁数、行数を指定する。

➔「桁数」は、1行の桁数を、40文字または80文字から選択して設定する。

➔「行数」は、1画面の行数を、20行または25行から選択して設定する。

➔「行数」は省略することもでき、省略した場合は現在設定されて

いる行数のままで変化しない。

➔書式②は、コミュニケーションポートに出力される文字のサイズを設定する。「サイズ」は、0から255まで（0は256と解釈される）指定することができる。指定された文字数をコミュニケーションポートに出力すると、改行コード（CR）を送出する。

➔書式③は、プリンタの1行に印字する文字数を設定する。「文字数」は、0から255まで設定でき、設定された文字数を印字すると改行する。

用例

```
100 CONSOLE 0,25,1,1
110 WIDTH 80,25
120 PRINT STRING$(255,"#")
130 PRINT "ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
140 KY$=INKEY$:IF KY$="" THEN 140
150 '--------------------
160 WIDTH 40,20
170 PRINT STRING$(255,"#")
180 PRINT "ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
190 KY$=INKEY$:IF KY$="" THEN 190
200 CLS
210 WIDTH 80,25
220 '--------------------
230 PRINT "ﾌﾟﾘﾝﾀ ﾉ ｼﾞｭﾝﾋﾞ ｶﾞ ﾃﾞｷﾀﾗ ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
240 KY$=INKEY$:IF KY$="" THEN 240
250 WIDTH LPRINT 60
260 LPRINT STRING$(255,"#")
```

CLS（クリアスクリーン）

■画面表示をクリア（消去）する

書式　CLS　機能番号

➔「機能番号」で指定された画面のクリア（消去）を行なう。

➔「機能番号」は、1～3の数値によって、次の画面をクリアする。省略値は1とみなされる。

1………テキスト画面だけをクリアする。クリアする範囲は、CONSOLE文で指定されたスクロールウィンドウ内だけである。

2………グラフィック画面だけをクリアする。クリアする範囲は、VIEW文によってビューポートとして指定された範囲だけである。

3………テキスト・グラフィックの両画面をクリアする。

➔テキスト画面のクリアの場合、CONSOLE文、COLOR文によって白黒、リバースモードに設定されていると、画面が白くなる。

➔グラフィック画面をクリアする場合、SCREEN文でカラーモードに設定されていると、COLOR文によって指定されたバックグラウンドカラーでクリアされる。

用例

```
100 CONSOLE 0,25,1,1
110 PRINT STRING$(160,"A")
120 PRINT "ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
130 KY$=INKEY$:IF KY$="" THEN 130
140 CLS
150 '-------------------------
160 SCREEN 3,0,0,1
170 PRINT STRING$(160,"B")
180 LINE (100,100)-(200,200),6,BF
190 PRINT "ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
```

```
200 KY$=INKEY$:IF KY$="" THEN 200
210 CLS 2
220 PRINT "ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
230 KY$=INKEY$:IF KY$="" THEN 230
240 '----------------------
250 PRINT STRING$(255,"#")
260 LINE (100,100)-(200,200),3,BF
270 PRINT "ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
280 KY$=INKEY$:IF KY$="" THEN 280
290 CLS 3
```

参照　CONSOLE／VIEW／COLOR／SCREEN

LOCATE（ロケート）

■カーソルを移動する

書式　① LOCATE　X座標, Y座標

② LOCATE　X座標, Y座標, カーソルスイッチ

➔テキスト画面のキャラクタ座標（「X座標」，「Y座標」）にカーソルを移動する。

➔キャラクタ座標は、画面の左上隅を（0，0）とする

➔「X座標」は水平方向の座標（桁に対応）を表わし、「Y座標」は垂直方向の座標（行に対応）を表わす。

➔「X座標」を省略した場合は0とみなされ、「Y座標」を省略した場合は現在カーソルが位置している行（Y座標）が指定されたとみなされる。

➔書式②は、「カーソルスイッチ」によってカーソルの表示状態を指定する。「X座標」、「Y座標」は省略することができる。

➔「カーソルスイッチ」は、0を指定するとカーソルが表示されなくなり、1を指定すると表示される。

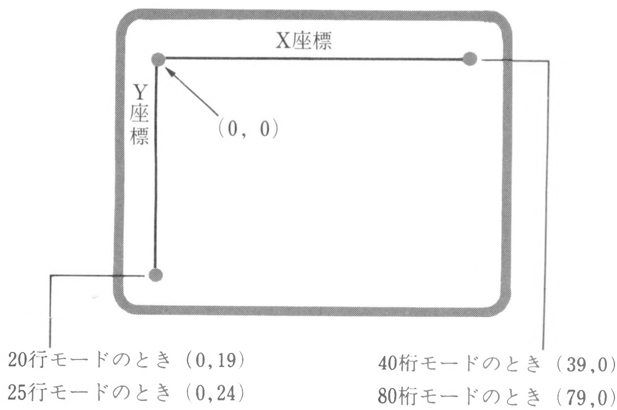
➔日本語文字列の表示、あるいは入力の際「X座標」が奇数桁の場

合、正しく表示、入力ができないので注意する。

用例

```
100 CONSOLE 0,25,1,1
110 LOCATE 20,10
120 PRINT "# <-ｺｺ ﾊ X=20,Y=10 ﾉ ｲﾁﾃﾞｽ"
130 PRINT:PRINT "ﾅﾆｶ ｷｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
140 KY$=INKEY$:IF KY$="" THEN 140
150 CLS
160 '---------------------
170 LOCATE ,,0
180 FOR I=0 TO 76
190   LOCATE I,12
200   PRINT" -->";
210   FOR J=1 TO 200:NEXT J
220 NEXT I
230 LOCATE ,,1
```

参照　CSRLIN／POS



CSRLIN（カーソルライン）

■現在カーソルの位置するY座標（行）を与える

書式　CSRLIN

➔テキスト画面上の、現在カーソルの位置するY座標（行）を与える関数。

➔キャラクタ画面の上端を0として、下に向かって数が大きくなる。

➔値の範囲は画面のモードにより、25行モードで0～24、20行モードで0～19となる。

用例

```
100 CONSOLE 0,25,1,1
110 LOCATE 0,22:INPUT "ｷﾞｮｳ ｦ ｼﾃｲｼﾃ ｸﾀﾞｻｲ (0-20)";Y
120 LOCATE 0,Y:PRINT "# ";CSRLIN;"ｷﾞｮｳ";
130 GOTO 110
```

参照　LOCATE／POS

POS (ポジション)

現在カーソルの位置するX座標(桁)を与える

書式　POS　(数式)

➔テキスト画面上の、現在カーソルの位置するX座標(桁)を与える関数。

➔キャラクタ画面の左端を0として、右に向かって数が大きくなる。

➔値の範囲は画面のモードにより、40行モードで0～39、80行モードで0～79となる。

➔「数式」はダミーで、どんな値を指定してもいいが、省略することはできない。一般に0を指定する。

用例

```
100 CONSOLE 0,25,1,1
110 LOCATE 0,22:INPUT "レツ ヲ シテイシテ クダサイ (0-20)";X
120 LOCATE X,10:PRINT "#"
130 LOCATE X,12:PRINT POS(0)
140 LOCATE X,13:PRINT " レ"
150 LOCATE X,14:PRINT " ツ"
160 GOTO 110
```

参照　LOCATE／CSRLIN

COLOR@（カラーアット）

■指定範囲の文字に色や機能を設定する

書式　COLOR@　(左上X座標，左上Y座標)－(右下X座標，右下Y座標)，ファンクションコード

➔テキスト画面上の「(左上X座標，左上Y座標)－（右下X座標，右下Y座標)」で指定される範囲の文字に、「ファンクションコード」で指定した色や機能を設定する。

➔範囲は左上の座標を「(左上X座標，左上Y座標)」、右下の座標を「(右下X座標，右下Y座標)」で指定し、その間を「―(マイナス)」で結ぶ。この2つの点を対角とする四角型の領域を範囲として指定する。

➔「ファンクションコード」の機能は、CONSOLE文で指定されている画面モードによって異なる。省略された場合は、7を指定したとみなされる。

➔白黒モードの場合（CONSOLE，，，　0）

0………ノーマル（通常の表示）
1………シークレット（文字は表示されない）
2………ブリンク（文字が点滅する）
3………シークレット（1に同じ）
4………リバース（文字色が反転表示される）
5………リバースシークレット（反転して文字は表示されない）
6………リバースブリンク（反転して文字が点滅する）
7………リバースシークレット（5と同じ）

➔カラーモードの場合（CONSOLE，，，　1）

0………黒
1………青

2………赤

3………紫（マゼンタ）

4………緑

5………水色（シアン）

6………黄色

7………白

➔ COLOR＠文による機能の設定は、表示されている文字に対してのみ有効であり、画面上になにも書かれていない場合は、なんの作用もない。また、設定された範囲の中に新たに書かれた文字などに対しては、この命令による影響は及ばない。

用　例

```
100 CONSOLE 0,25,0,1
110 CLS
120 FOR Y=0 TO 20
130  PRINT STRING$(80,"#");
140 NEXT Y
150 LOCATE 0,21:INPUT "ﾋﾀﾞﾘ ｳｴ ｲﾁ 'X' ｦ ｼﾃｲｼﾃ ｸﾀﾞｻｲ
    (0-78)";X1
160 LOCATE 0,22:INPUT "ﾋﾀﾞﾘ ｳｴ ｲﾁ 'Y' ｦ ｼﾃｲｼﾃ ｸﾀﾞｻｲ
    (0-18)";Y1
170 LOCATE 0,21:INPUT "ﾐｷﾞ ｼﾀ  ｲﾁ 'X' ｦ ｼﾃｲｼﾃ ｸﾀﾞｻｲ
    (1-79)";X2
180 LOCATE 0,22:INPUT "ﾐｷﾞ ｼﾀ  ｲﾁ 'Y' ｦ ｼﾃｲｼﾃ ｸﾀﾞｻｲ
    (1-20)";Y2
190 COLOR@ (X1,Y1)-(X2,Y2),2
```

参　照　COLOR／CONSOLE

メニューを反転して自由な位置に表示する

```
100 '************************************
110 '       リバースカーソル選択
120 '************************************
130 CONSOLE 0,24,1,0
140 WIDTH 80,25
150 CLS
160 MX=10:X=0:Y=2
170 DIM MN$(MX) ――――― メニューの文字の入る配列変数の宣言
180 '------------------------------------
190 '               MAIN
200 '------------------------------------
210 LOCATE 0,0:PRINT "reverse cursole selection"
220 MX=MX-1
230 'ﾒﾆｭｰ ﾋｮｳｼﾞ ------------------------
240 FOR I=0 TO MX                      ┌―― モデルのためにメ
250   MN$(I)="select No."+STR$(I)        ニュー番号を設定
260   LOCATE X,Y+I:PRINT MN$(I)
270 NEXT I
280 SX=X:SY=Y:SLN=0                          リバース表示
290 COLOR@ (SX,SY)-(SX+LEN(MN$(1)),SY),4 ――┘
300 'ﾘﾊﾞｰｽｶｰｿﾙ ｾﾝﾀｸ --------------------
310 SN=-1
320 WHILE SN<0
330   LX=SX:LY=SY
340   AN$=""
350   WHILE AN$="" ――――┐
360      AN$=INKEY$     ├― キー入力待ち
370   WEND ―――――――――――┘
380   IF AN$=CHR$(&H1E) THEN SY=SY-1:IF SY<Y
      THEN SY=Y      └―― カーソル上の場合
390   IF AN$=CHR$(&H1F) THEN SY=SY+1:IF Y+MX<SY
      THEN SY=Y+MX    └― カーソル下の場合
400   COLOR@ (LX,LY)-(LX+LEN(MN$(1)),LY),0
410   COLOR@ (SX,SY)-(SX+LEN(MN$(1)),SY),4
420   IF AN$=CHR$(&HD) THEN SN=SY-Y
430 WEND          └―――リターンキーの場合
440 '------------------------------------
450 LOCATE 0,20:PRINT "you get [";MN$(SN);"]"
460 END                     選択された番号 ―――――┘
```

●実用プログラムに応用すると便利

画面にメニューを表示して、カーソルキーで反転表示を移動し、⏎キーで選択するプログラムである。160行の値の設定を変えると、画面の大きさの範囲内で任意の位置にメニューを表示させることができる。

MX＝メニューの最大値（個数）
X　＝メニューの始まるX座標
Y　＝メニューの始まるY座標
MN$(　)＝メニューの文字列

450行以降に、実際に選択されたメニューによる処理のプログラムを置く。

いくつかの処理をメニューから選択して実行する場合などに、番号やアルファベットで入力するのではなく、カーソルキーと⏎キーだけで選択できるという便利さがある。

MN$(　)に、処理を表わす適当な文字列をセットすれば、自分で作ったいろいろなプログラムの中で応用できる。



6. グラフィック&サウンド制御

SCREEN（スクリーン）

グラフィック画面に対するモードを設定する

書式 SCREEN　画面モード，画面スイッチ，アクティブスイッチ，ディスプレイスイッチ

➔グラフィック画面をカラーモードにするか、白黒モードにするかなどの種々のモードを設定する。

➔「画面モード」は、次の0～3で基本的モードを設定する。

0………カラー（640×200ドット、4ページ）

1………白黒　（640×200ドット、12ページ）

2………高分解能白黒（640×400ドット、6ページ）

3………高分解能カラー（640×400ドット、2ページ）

➔「画面スイッチ」は、現在表示されているグラフィック画面を一時的に画面から消したり、表示したりするスイッチである。

0………グラフィックを画面に表示する。通常0を指定。

2………グラフィックを一時的に画面から消去する。

➔「アクティブスイッチ」、「ディスプレイスイッチ」は、「画面モード」によって設定されたグラフィックのページのうちから、書き込みを行なうページと、画面に表示するページを指定する。ページ指定には多数の指定パターンがあるので、パソコンに付属のBASICリファレンスマニュアルを参照のこと。

➔一般的に「SCREEN　3, 0, 」（640×400ドット、2ページ）に続

けて指定した場合、次の指定が可能である。

アクティブスイッチ	0………	1ページに書き込み
	1………	2ページに書き込み
ディスプレイスイッチ	0………	全ページ表示しない
	1………	1ページのみ表示
	17………	2ページのみ表示

用 例

```
100 SCREEN 1,0,0,0
110 GOSUB *GRPH
120 PRINT "「白黒モード、表示しない」で書き込み"
130 GOSUB *KYIN
140 SCREEN 1,0,0,7
150 PRINT "「白黒モード、全ページ表示」に指定"
160 GOSUB *KYIN
170 SCREEN 3,0,0,1
180 PRINT "「高分解能カラーモード、1ページ表示」に指定"
190 GOSUB *KYIN
200 SCREEN 3,0,0,17
210 PRINT "「高分解能カラーモード、2ページ表示」に指定"
220 GOSUB *KYIN
230 SCREEN 3,2,0,1
240 PRINT "「高分解能カラーモード、1ページ表示しない」
 に指定"
250 GOSUB *KYIN
260 SCREEN 3,0,0,1
270 PRINT "「高分解能カラーモード、1ページ表示」に指定"
280 GOSUB *KYIN
290 CLS 3
300 END
310 *GRPH
320   LINE (100,100)-(200,150),5,BF
330   RETURN
340 *KYIN
350   KY$=INKEY$:IF KY$="" THEN 350
360   RETURN
```

参 照 COLOR

WINDOW（ウィンドウ）

■ワールド座標系内の表示領域を指定する

書式　WINDOW　（左上X，左上Y）－（右下X，右下Y）

➔「ワールド座標系」とは、X座標、Y座標とも-1.70141×10^{38}～1.70141×10^{38} の範囲の値をとることができる論理的なグラフィック座標で、LINEやCIRCLEなどのグラフィック命令は、この値の範囲で座標を指定することができる。

➔ WINDOW文は、このワールド座標系のうち、どの部分（領域）をディスプレイ画面に表示させるかを指定する。

➔領域の指定は、ワールド座標系のX、Y座標の値を使って、領域の左上隅の座標を「(左上X，左上Y)」で、右下隅の座標を「(右下X，右下Y)」で指定して、「－（マイナス）」で結んで指定する。

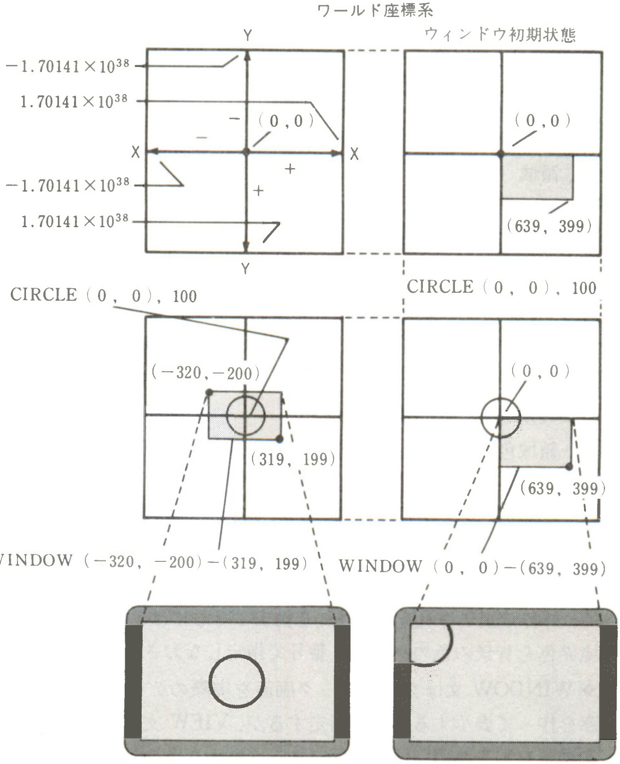
➔ WINDOW文で指定された領域は、ちょうどワールド座標系の世界の一部を、窓から望遠鏡を通して見ているように、グラフィック画面に表示される。

さらにWINDOW文で指定された範囲は、ビューポートという小窓に投影されて、実際のディスプレイに表示される。

用例

```
100 SCREEN 3,0,0,1
110 WINDOW (0,0)-(639,399)
120 CLS 3
130 CIRCLE (0,0),100
140 KY$=INKEY$:IF KY$="" THEN 140
150 WINDOW (-320,-200)-(319,199)
160 CLS 3
170 CIRCLE (0,0),100
180 KY$=INKEY$:IF KY$="" THEN 180
190 CLS 3
200 WINDOW (0,0)-(639,399)
210 END
```

左ページ用例のプログラムから、ワールド座標系とディスプレイ画面表示の関係を図説すると、次のようになる。



ディスプレイ画面表示

参照 VIEW／SCREEN

VIEW（ビュー）

■ディスプレイ画面上の表示領域を指定する

書式　VIEW　(左上X，左上Y)－(右下X，右下Y)，領域色，境界色

➔グラフィック画面上の図形を、実際にディスプレイ画面上に表示する領域（ビューポートという）を指定する。

➔領域はオリジナルスクリーン座標で指定する。この座標は、ディスプレイ画面の左上を（0，0）とする、画面のドットに1対1で対応する物理的なX、Y座標で、640×400ドットの画面の場合、Xの最大値は639、Yの最大値は399である。この範囲の値を使って、領域の左上隅の座標を「(左上X，左上Y)」で、右下隅の座標を「(右下X，右下Y)」で指定し、「－（マイナス)」で結んで指定する。

この領域をビューポートという。

➔「領域色」をパレット番号で指定すると、指定された色でビューポート内を塗りつぶすことができる。

➔「境界色」をパレット番号で指定すると、指定された色でビューポートの外枠を描くことができる。

➔CLS文によるグラフィックの消去範囲は、ビューポート内だけで、外枠は消去されない。外枠を消去させるには、再度VIEW文で境界色を背景の色のパレット番号で指定しなおさなければならない。

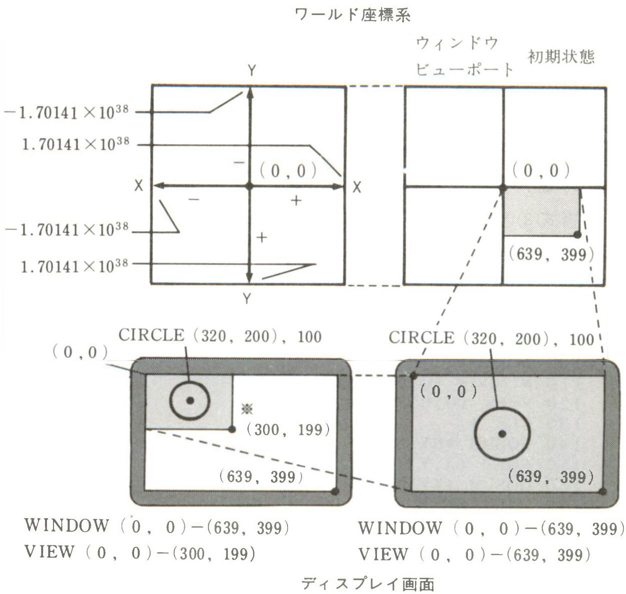
➔WINDOW文はグラフィック画面を実際のディスプレイ上に小窓を作って表示する領域を指定するが、VIEW文はこの位置を決める働きをするため、同じグラフィックを大きさや位置を変えて、同一画面上に表示させることもできる。

参照　WINDOW／CLS／SCREEN

用 例

```
100 SCREEN 3,0,0,1
110 WINDOW (0,0)-(639,399)
120 VIEW (0,0)-(639,399)
130 CLS 3
140 CIRCLE (320,200),100
150 KY$=INKEY$:IF KY$="" THEN 150
160 CLS 3
170 VIEW (0,0)-(300,199)
180 CIRCLE (320,100),100
190 KY$=INKEY$:IF KY$="" THEN 190
200 CLS 3
210 VIEW (0,0)-(639,399)
220 END
```

※図中の(300, 199)はビューポートを表示しているディスプレイ画面の座標。小窓を通して右側の図を見ているので、見かけ上同じ大きさで指定した円が小さく見える。



PSET／PRESET
(ポイントセット／ポイントリセット)

■グラフィック画面にドットを表示／消去する

書　式

① PSET　(X, Y)，パレット番号

② PRESET　(X, Y)

③ PSET　STEP　(X, Y)，パレット番号

④ PRESET　STEP　(X, Y)

➔書式① PSET 文は、ワールド座標系上で指定された「(X，Y)」の位置に、「パレット番号」で指定した色でドットを表示する。

➔「パレット番号」を省略すると、COLOR 文で指定されているフォアグラウンドカラーが指定されたとみなす。

➔書式② PRESET文は、ワールド座標系上で指定された「(X，Y)」の位置に表示されているドットを消去する。(COLOR 文で指定されているバックグラウンドカラーで、点を塗りかえる働きをする。)

➔書式③と④は、「STEP」を付加することによって現在の LP(Last Referenced　Point＝最後にグラフィック操作の行なわれた座標)から相対的な座標（X，Y方向に移動するドット数）で指定する。

用　例

```
100 SCREEN 3,0,0,1
110 WINDOW (0,0)-(639,399)
120 VIEW (0,0)-(639,399)
130 CLS 3
140 GOSUB *KYIN
150 PSET (X,Y),C
160 GOSUB *KYIN
170 PSET STEP (X,Y),C
180 GOSUB *KYIN
190 PRESET (X,Y)
200 GOSUB *KYIN
```

```
210 PRESET STEP (X,Y)
220 CLS 3
230 END
240 *KYIN
250   INPUT "     X = ";X
260   INPUT "     Y = ";Y
270   INPUT "COLOR = ";C
280   RETURN
```

参照　COLOR／PAINT

LINE（ライン）

■2点間を結ぶ直線を引いたり四角形を描く

書式
① LINE　(始点X，始点Y)－(終点X，終点Y)，パレット番号
② LINE　(始点X，始点Y)－(終点X，終点Y)，パレット番号,,ラインスタイル
③ LINE　(始点X，始点Y)－(終点X，終点Y)，パレット番号，B
④ LINE　(始点X，始点Y)－(終点X，終点Y)，パレット番号，BF，塗りつぶしパレット番号

➔書式①は、ワールド座標系上で指定された「(始点X，始点Y)」と「(終点X，終点Y)」の間を結ぶ直線を、「パレット番号」で指定した色で描く。

➔書式②では、「，ラインスタイル」を16進数の数値で追加指定して、直線の形態を点線や破線などにすることができる。

➔書式③は、「,B(Box＝箱の意味)」を追加指定すると、ワールド座標系上で指定された「(始点X，始点Y)」と「(終点X，終点Y)」を、それぞれ左上隅、右下隅の2点とし、それを対角とする四角形の

箱を描く。

➔書式④では、「,BF(Box fill)」を追加指定すると、四角形の中を「塗りつぶしパレット番号」で指定した色で塗りつぶす。

用例

```
100 SCREEN 3,0,0,1
110 WINDOW (0,0)-(639,399)
120 VIEW (0,0)-(639,399)
130 CLS 3
140 LINE (0,0)-(100,100),7
150 LINE (0,200)-(500,200),1,,&HAAAA
160 LINE (0,220)-(500,220),2,,&HCCCC
170 LINE (0,240)-(500,240),3,,&HF0F0
180 LINE (0,260)-(500,260),4,,&HC2C2
190 LINE (0,280)-(500,280),5,,&HFCFC
200 LINE (0,300)-(500,300),6,,&HFF18
210 LINE (0,320)-(500,320),7,,&HFFF6
220 LINE (150,0)-(250,100),6,B
230 LINE (300,0)-(400,100),2,BF,5
240 KY$=INKEY$:IF KY$="" THEN 240
250 CLS 3
260 END
```

参照　COLOR／PAINT

●LINE　**応用プログラム**

データ数を増やして実用プログラムに組み込む

```
100 '**************************************
110 '              棒グラフ
120 '**************************************
130 CONSOLE 0,25,0,1        ┐
140 WIDTH 80,25             │
150 SCREEN 3,0,0,1          ├ グラフィック画面の
160 WINDOW (0,0)-(639,399)  │ 初期設定
170 VIEW (0,0)-(639,399)    ┘
180 CLS 3
190 DIM DT(15)              ┐
```

```
200 FOR I=1 TO 15          ─グラフ化するデータを配列変数に読み込む
210   READ DT(I)
220 NEXT I
230 '---------------------------------------
240                 *MAIN
250 '---------------------------------------
260 LOCATE 30,0:PRINT "ﾎﾞ    ｳ    ｸﾞ    ﾗ    ﾌ"
270 FOR Y=1 TO 21 STEP 2
280   LINE (75,Y*16-1)-(639,Y*16-1),4      ─グラフの目盛り
290 NEXT Y                                   Y軸
300 FOR Y=0 TO 20 STEP 2
310   LOCATE 5,Y:PRINT 100-(Y*5)
320 NEXT Y
330 LINE (79,16)-(79,335),4                ── X軸
340 '---------------------------------------
350 FOR I=1 TO 15
360   C=INT(I MOD 7)+1 ──グラフの表示色の設定
370   LINE (I*32+95,336-(DT(I)*3.2))-(I*32+109,335),
C,BF                       └──棒グラフの表示
380   LOCATE I*4+12,21:PRINT CHR$(I+&H40);
390 NEXT I                 └──データ番号の表示
400 '---------------------------------------
410 KY$=INKEY$:IF KY$="" THEN 410
420 CLS 3
430 END
440 DATA 20,36,32,54,75,60,32,42,58,71,88,84,79,68,
72                         └──グラフ化するデータ
```

● 「棒グラフ」プログラムの使い方

RUNすると、15件のデータが棒グラフになって表示される。

データの数は、190行の配列宣言「DT（15）」で決まっているが、必要ならば370行のX座標を変更して棒グラフの幅を小さくして増やすこともできる。

データは、440行のデータ文にある。実際のデータをデータ文としておくことによって、実用プログラムに組み込むこともできる。

CIRCLE（サークル）

■円やだ円を描く

書式

① CIRCLE （中心X，中心Y），半径，パレット番号

② CIRCLE （中心X，中心Y），半径，パレット番号，開始角度，終了角度

③ CIRCLE （中心X，中心Y），半径，パレット番号，開始角度，終了角度，F，塗りつぶしパレット番号

④ CIRCLE （中心X，中心Y），半径，パレット番号，開始角度，終了角度，比率，F，塗りつぶしパレット番号

➔書式①は、ワールド座標系上で指定された「(中心X，中心Y)」を中心とし、「半径」で指定される大きさの円を、「パレット番号」で指定された色で描く。

➔書式②は、「開始角度」、「終了角度」で指定した範囲に円弧を描く。角度の指定単位はラジアン角で、-2π から $+2\pi$ の範囲で指定し、省略した場合は、それぞれ0と2π とみなされる。

➔書式③では、「F(Fill＝満たすという意味)」を追加指定して、描いた円や円弧の中を、「塗りつぶしパレット番号」で指定した色で塗りつぶすこともできる。

➔書式④では、「比率」を追加指定して、だ円を描くこともできる。「比率」は、(垂直方向の半径)／(水平方向の半径)で、画面上のドット数に対応する数値で指定する（640×200ドットの場合は½の比率になる)。

用例

```
100 SCREEN 3,0,0,1
110 WINDOW (0,0)-(639,399)
120 VIEW (0,0)-(639,399)
130 CLS 3
140 PAI=3.1415
150 CIRCLE (100,100),40,2
160 CIRCLE (100,200),40,3,0,PAI
170 CIRCLE (100,300),40,4,0,3*PAI/2,,F,5
180 CIRCLE (300,100),40,5,0,,50/100
190 CIRCLE (300,200),40,6,0,,100/50
200 CIRCLE (300,300),40,2,0,3*PAI/2,50/100,F,5
210 KY$=INKEY$:IF KY$="" THEN 210
220 CLS 3
230 END
```

CIRCLE ―――――――――――――――――――――――――――――― 応用プログラム

データ数を増減するときにはカラーの色指定に注意

```
100 '*************************************
110 '              円グラフ
120 '*************************************
130 CONSOLE 0,25,0,1           ┐
140 WIDTH 80,25                │
150 SCREEN 3,0,0,1             ├ グラフィック画面の初期設定
160 WINDOW (0,0)-(639,399)     │
170 VIEW (0,0)-(639,399)       │
180 CLS 3                      ┘
190 DIM DT(7),PT(7)
200 PAI=3.1415 ―――――――――― πの値を設定
210 FOR I=1 TO 7   ┐
220   READ DT(I)   ├ グラフ化するデータの読み込み
230 NEXT I         ┘
240 '-------------------------------------
250                *MAIN
260 '-------------------------------------
270 LOCATE 23,0:PRINT "ｴ ﾝ ｸﾞ ﾗ ﾌ"
280 FOR Y=1 TO 7
290   COLOR Y
```

```
300    LOCATE 60,Y*2+6:PRINT CHR$(Y+&H40);
310 NEXT Y                          データ番号の色を表示
320 COLOR 7
330 '----------------------------------------
340 TL=0
350 FOR I=1 TO 7
360    TL=TL+DT(I)                   データの合計を求める
370 NEXT I
380 FOR I=1 TO 7
390    PT(I)=DT(I)/TL                各データの百分率を求める
400 NEXT I
410 ST=PAI/2                         グラフの表示開始位置
420 FOR I=7 TO 1 STEP -1
430    EN=(ST+PT(I)*PAI*2)
440    IF PAI*2<EN THEN EN=EN-PAI*2                   ※グラフ
450    CIRCLE (240,192),160,I,-ST,-EN,,F,I             表示
460    ST=EN
470 NEXT I
480 FOR Y=1 TO 7
490    LOCATE 64,Y*2+6:PRINT DT(Y);             データの
500    PT(Y)=INT(PT(Y)*10000)/100               百分率表示
510    LOCATE 70,Y*2+6:PRINT USING "##.##%";PT(Y)
520 NEXT Y
530 '----------------------------------------
540 KY$=INKEY$:IF KY$="" THEN 540
550 CLS 3
560 END                              グラフ化するデータ
570 DATA 840,690,640,510,430,210,200
```

● 「円グラフ」プログラムの使い方

RUNすると、7件のデータが円グラフになって表示される。

データの数は、190行の配列宣言「DT（7），PT（7）」で決まっているが、必要ならばFOR～NEXT文のパラメータを変更することによって数を増減することもできる。このとき、カラーの色指定に注意する。

変数TLには、データの合計が加算される。これは、データの百分率を求めるための分母を得るためである。大きな値のデータを扱うときは、倍精度実数にするとよい。

COLOR／COLOR＝
（カラー／カラーイコール）

■文字やグラフィックの表示色、カラーパレットを設定

書式　①　COLOR　文字色，バックグラウンドカラー，ボーダーカラー，フォアグラウンドカラー，パレットモード

②　COLOR　ファンクションコード

③　COLOR＝(パレット番号，カラーコード)

④　COLOR

➔書式①は、テキスト画面の文字の色やグラフィックの表示色を設定する。

➔「文字色」は、次のカラーコードによって、キャラクタ画面の文字の色を指定する。

0……黒　　1……青　　2……赤　　3……紫（マゼンタ）

4……緑　　5……水色（シアン）　　6……黄色　　7……白

➔「バックグラウンドカラー」は、グラフィック画面のバック（地）をパレットコードで指定する。

➔「ボーダーカラー」は、ディスプレイ画面のうち、BASICで使用していない部分、つまり何も表示されない外側の余白の部分の色をカラーコードで指定する。専用高解像度ディスプレイでは、指定しても無視される。

➔「フォアグラウンドカラー」は、グラフィック画面に点や線などを表示するときに使う色をパレットコードで指定する。

➔「パレットモード」は、グラフィック画面に対する色指定のモードを、次の3つのうちから指定する。

0………8色中8色モード

1………4096色中8色モード

2………4096色中16色モード

➔書式②は、CONSOLE文で白黒モードが設定されている（CONSOLE，，，0）場合のテキスト画面の文字に、次の番号によって種々の機能を設定する。

0………ノーマル（通常の表示）

1………シークレット（文字を表示しない）

2………ブリンク（文字が点滅する）

3………シークレット（1に同じ）

4………リバース（文字が反転表示される）

5………リバースシークレット（反転して文字が表示されない）

6………リバースブリンク（反転して文字が点滅する）

7………リバースシークレット（5に同じ）

➔書式③は、パレット番号に対応する色を、カラーコードで設定する。

➔「パレット番号」は、8色モードで0～7、16色モードで0～15のカラーパレットに付けられた固有の番号で、これに対応する色を「カラー番号」で設定する。

➔カラーパレットに対応するカラーコードを変更すると、表示されているグラフィックの絵の色を一瞬にして変えることができる。

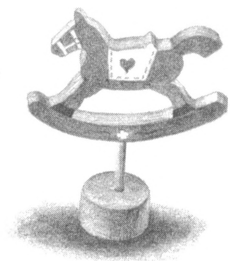
➔書式④では、パレット番号とカラー番号の対応を、初期状態に戻すことができる。

用　例

```
100 CONSOLE 0,25,0,1
110 SCREEN 3,0,0,1
120 CONSOLE 0,25,0,1
130 FOR I=0 TO 7
140   COLOR I
150   PRINT "ﾊﾟﾚｯﾄ ｺｰﾄﾞ ﾊ ";I;"ﾊﾞﾝ ﾃﾞｽ"
```

```
160 NEXT I
170 KY$=INKEY$:IF KY$="" THEN 170
180 CONSOLE 0,25,0,0
190 FOR I=0 TO 7
200   COLOR I
210   PRINT "ﾌｧﾝｸｼｮﾝ ｺｰﾄﾞ ﾊ ";I;"ﾊﾞﾝ ﾃﾞｽ"
220 NEXT I
230 COLOR 0
240 KY$=INKEY$:IF KY$="" THEN 240
250 CONSOLE 0,25,0,1
260 COLOR=(1,7)
270 COLOR=(2,6)
280 COLOR 1:PRINT "ﾊﾟﾚｯﾄ ｺｰﾄﾞ ﾊ  1  ﾊﾞﾝ ﾃﾞｽ"
290 LINE (200,8)-(600,8),1
300 COLOR 2:PRINT "ﾊﾟﾚｯﾄ ｺｰﾄﾞ ﾊ  2  ﾊﾞﾝ ﾃﾞｽ"
310 LINE (200,24)-(600,24),2
320 KY$=INKEY$:IF KY$="" THEN 320
330 COLOR
340 COLOR 7
350 CLS 3
360 END
```

参照　CONSOLE



PAINT（ペイント）

■境界色で囲まれた領域を指定色で塗りつぶす

書式　① PAINT　(始点X，始点Y)，領域色，境界色
② PAINT　STEP　(始点X，始点Y)，領域色，境界色

➔ワールド座標系上で指定された、「(始点X，始点Y)」を含む「境界色」で囲まれた領域を、「領域色」で指定された色で塗りつぶす。

➔「境界色」はパレット番号で指定し、省略された場合は「領域色」と同じとみなされる。

➔「領域色」はパレット番号で指定し、省略された場合はCOLOR文で指定されたフォアグラウンドカラーが指定されたとみなされる。

➔「(始点X，始点Y)」で指定された点が「境界色」と同じ色であった場合は、PAINT文は何もしないので注意する。

➔ PAINT文はビューポート内のみで働くので、ビューポートの境界は、PAINT文の動作の境界となる。

用例

```
100 CONSOLE 0,25,0,1
110 SCREEN 3,0,0,1
120 CONSOLE 0,25,0,1
130 CIRCLE (350,50),50,7
140 LINE (300,150)-(400,250),7,B
150 FOR I=0 TO 7
160   PRINT "ﾊﾟﾚｯﾄ ｺｰﾄﾞ ﾊ ";I;"ﾊﾞﾝ ﾃﾞｽ"
170   PAINT (350,50),I,7
180   PAINT STEP (0,150),I,7
190   KY$=INKEY$:IF KY$="" THEN 190
200 NEXT I
210 KY$=INKEY$:IF KY$="" THEN 210
220 CLS 3
230 END
```

参照　COLOR

GET@（ゲットアット）

■グラフィックパターンを配列変数に取り込む

書式　GET@（左上X，左上Y）－（右下X，右下Y），配列変数名（添え字番号）

➔画面上のスクリーン座標（ビューポートに展開されている論理座標）上の2点、（左上X，左上Y）と（右下X，右下Y）間を対角とする四角形の領域に描かれている絵を、グラフィックパターンのデータとして、「配列変数名」で指定された配列変数に取り込む。

➔「配列変数名」は、データを取り込むための数値型の配列変数をユーザーが事前に配列宣言をして、用意しなければならない。

➔データを取り込む際、「添え字番号」で、配列変数のどの添え字番号の要素から始めるかを指定することができる。一般に省略するが、省略した場合は、配列変数のいちばん初めから取り込まれる。

➔データを取り込む配列変数は、データのバイト数に応じて必要な大きさで配列宣言をしておくが、その大きさの計算は次のとおりである。

添え字の値………((((右下X－左上X)＋8)¥8)×(((右下Y－左上Y)＋1)×M＋4))¥N＋1

ただし、Mは、白黒モードのとき………M＝1
8色モードのとき………M＝3
16色モードのとき………M＝4
Nは、整数型配列の場合………N＝2
単精度型配列の場合……N＝4
倍精度型配列の場合……N＝8

➔GET@文で取り込んだデータは、PUT@文で画面に表示する。

用例

```
100 CONSOLE 0,25,0,1
110 SCREEN 3,0,0,1
120 CONSOLE 0,25,0,1
130 LX=0:LY=0:RX=50:RY=50
140 M=3
150 N=2
160 DM=((((RX-LX)+8)¥8)*(((RY-LY)+1)*M+4))¥N+1
170 DIM GD%(DM)
180 '--------------------------------------
190 LINE (0,0)-(50,50),7,BF
200 CIRCLE (25,25),15,2,,,,F,2
210 KY$=INKEY$:IF KY$="" THEN 210
220 '--------------------------------------
230 GET@ (0,0)-(50,50),GD%
240 '--------------------------------------
250 PUT@ (200,200),GD%
260 KY$=INKEY$:IF KY$="" THEN 260
270 CLS 3
280 END
```

参照　PUT@／VIEW／DIM

PUT@(プットアット)

■グラフィックパターンを画面に表示する

書式　PUT@　(左上X，左上Y)，配列変数名(添え字番号，機能，フォアグラウンドカラー，バックグラウンドカラー

➔ GET@文で配列に取り込んだグラフィックパターンを画面上の指定した位置に表示する。

➔位置の指定は、「(左上X，左上Y)」によって、画面上のスクリーン座標（ビューポートに展開されている論理座標）で指定する。

➔「配列変数名」は、表示したいグラフィックパターンのデータが

格納されている配列の変数名を指定する。

➔「添え字番号」は、配列内のどこからデータを取り始めるかを指定する。省略した場合、配列の先頭の要素からデータを取り始める。

➔「機能」は、グラフィックパターンを画面に表示する際に、種々の機能を指定して表示するもので、以下のものがある。

PSET …… グラフィックパターンをそのまま表示する。

PRESET…各ドットのパレット番号を「7−そのドットのパレット番号」(8色モード)、「15−そのドットのパレット番号」(16色モード) として表示する。

OR ……… 画面上のグラフィックパターンとドットごとにOR (論理和) を取って表示する。

AND …… 画面上のグラフィックパターンとドットごとにAND (論理積) を取って表示する。

XOR …… 画面上のグラフィックパターンとドットごとにXOR (排他的論理和) を取って表示する。

➔「機能」が省略されると、XORとみなされる。

➔「フォアグラウンドカラー，バックグラウンドカラー」は必ずペアで使い、白黒モードで取り込んだグラフィックパターンをカラーモードで表示するときに、色を指定するのに使う。

「フォアグラウンドカラー」には取り込んだときに白であったドットにつける色を、「バックグラウンドカラー」には黒であったドットにつける色を指定する。

用例

```
100 CONSOLE 0,25,0,1
110 SCREEN 3,0,0,1
120 CONSOLE 0,25,0,1
130 CLS 3
140 LX=0:LY=0:RX=50:RY=50
150 M=3
```

```
160 N=2
170 DM=((((RX-LX)+8)¥8)*(((RY-LY)+1)*M+4))¥N+1
180 DIM GD%(DM)
190 '-------------------------------------
200 LINE (0,0)-(50,50),7,BF
210 CIRCLE (25,25),15,2,,,,F,2
220 '-------------------------------------
230 GET@ (0,0)-(50,50),GD%
240 '-------------------------------------
250 LOCATE 10,8:PRINT "ｼﾊﾞﾗｸ ｵﾏﾁ ｸﾀﾞｻｲ"
260 LINE (0,150)-(300,300),6,BF
270 PUT@ (0,200),GD%,PSET
280 PUT@ (50,200),GD%(150),PSET
290 PUT@ (100,200),GD%,PRESET
300 PUT@ (150,200),GD%,OR
310 PUT@ (200,200),GD%,AND
320 PUT@ (250,200),GD%,XOR
330 KY$=INKEY$:IF KY$="" THEN 330
340 CLS 3
350 END
```

参照　GET@／VIEW

ROLL（ロール）

■グラフィック画面を上下左右にスクロールする

書式　① ROLL　上下方向ドット数，左右方向ドット数

② ROLL　上下方向ドット数，左右方向ドット数，Y または N

➔グラフィック画面を「上下方向ドット数」、「左右方向ドット数」で指定されたドット数分だけ、上下左右にスクロールする。

➔「上下方向ドット数」は、画面の縦方向のスクロール量を指定する。

640×200ドットの画面では、−199から199の間

640×400ドットの画面では、－399から399までの間の値で指定する。

正の値の場合は上方向に、負の場合は下方向にスクロールする。

省略した場合は、上下方向にスクロールしない。

➔「左右方向ドット数」は、横方向のスクロール量を指定する。

－639から639までの間の値で指定し、正の値の場合は右方向に、負の場合は左方向にスクロールする。

スクロールは、指定されたドット数以下でもっとも大きい8の倍数分だけ行なわれる。

省略した場合は、左右方向にスクロールしない。

➔書式②では、「ＹまたはＮ」を付加することができる。「Ｙ」を指定するとバックグラウンドカラーで、「Ｎ」を指定または省略するとパレット番号0の色で、スクロールによって新しく現れた領域をクリアする。

用例

```
100 CONSOLE 0,25,0,1
110 SCREEN 3,0,0,1
120 CONSOLE 0,25,0,1
130 CLS 3
140 LINE (300,200)-(350,250),7,BF
150 CIRCLE (325,225),15,2,,,,F,2
160 INPUT " X = ";X
170 INPUT " Y = ";Y
180 ROLL X,Y
190 GOTO 160
200 END
```

BEEP (ビープ)

■内蔵スピーカを鳴らす

書式　① BEEP

　　　② BEEP　0または1

➔ 書式①は、内蔵のスピーカを一瞬「ピッ」と鳴らす。

➔ 書式②では、スピーカを鳴らしたり止めたりすることができる。「1」はスイッチONの状態で、スピーカを鳴らす。「0」はスイッチOFFの状態で、スピーカを止める。

用例

```
100 CONSOLE 0,25,0,1
110 BEEP
120 INPUT "ON =";N
130 INPUT "OFF=";F
140 PRINT
150 FOR I=1 TO N
160   BEEP 1
170   FOR J=1 TO F
180     BEEP 0
190   NEXT J
200 NEXT I
210 GOTO 110
220 END
```



7. ファイル制御

CHAIN（チェイン）

ディスク上のプログラムを読み込んで実行する

書式

① CHAIN　"ドライブ名：ファイル名"

② CHAIN　"ドライブ名：ファイル名"，開始行番号，ALL

③ CHAIN　MERGE　"ドライブ名：ファイル名"，開始行番号，ALL，DELETE　削除開始行番号－削除終了行番号

➔書式①は、現在実行中のプログラムを消去して、「"ドライブ名：ファイル名"」で指定されたディスク上のプログラムを読み込んで実行する。

➔書式②は、読み込まれたプログラムの、実行を開始する行番号を「開始行番号」で指定する。開始行番号はラベルを使用することはできない。「ALL」は、読み込まれたプログラムへ、変数や配列の内容を引き渡す場合に指定する。

➔書式③は、現在実行中のプログラムと読み込むプログラムを連結（マージ＝合併）して、指定された「開始行番号」（省略された場合は先頭の行）から実行を開始する。この場合、読み込まれるプログラムはアスキーセーブされていなければならない。

➔「DELETE　削除開始行番号－削除終了行番号」は、ディスク上のプログラムを読み込むのに先だって、現在実行中のプログラムの、指定された行番号の範囲を削除する。削除しない場合は、省略する。

➔ 書式③では、連結されたプログラムでもDEFINT、DEFSNG、DEFDBL、DEFSTRなどの宣言が有効だが、書式②では無効なので、必要な場合は読み込まれるプログラムで宣言しなければならない。

➔ プログラムを連結すると、DEF FNおよびON KEY GOSUB、ON ERROR GOTOなどの例外処理命令の制御情報が無効になるので、必要な場合は読み込まれるプログラムで宣言しなければならない。

用例

・プログラム1

```
100 CONSOLE 0,25,1,1
110 INPUT "ｽｳﾁ A=";A
120 INPUT "ｽｳﾁ B=";B
130 INPUT "ｹｲｻﾝ(+ - * /)";C$
140 IF C$="+" THEN 190
150 IF C$="-" THEN 200
160 IF C$="*" THEN 210
170 IF C$="/" THEN 220
180 GOTO 110
190 CHAIN "2:CHAIN2.BAS",200,ALL
200 CHAIN "2:CHAIN2.BAS",210,ALL
210 CHAIN "2:CHAIN2.BAS",220,ALL
220 CHAIN "2:CHAIN2.BAS",230,ALL
230 END
```

・プログラム2（1から呼び出される。ディスクにセーブしておく）

```
100 PRINT "ｺﾉ ｷﾞｮｳ ﾊ ﾋｮｳｼﾞ ｻﾚﾏｾﾝ"
110 '
200 PRINT A+B :GOTO 240
210 PRINT A-B :GOTO 240
220 PRINT A*B :GOTO 240
230 PRINT A/B :GOTO 240
240 PRINT
250 PRINT "A=";A
260 PRINT "B=";B
270 PRINT "C$= ";C$
280 END
```

参照 MERGE／COMMON／SAVE

COMMON（コモン）

■連結されたプログラムに引き渡す変数を定義する

書式　COMMON　変数名，変数名，………

➔ CHAIN文によってディスク上のプログラムが連結された場合、次のプログラムで一部の変数を引き続き使用するために、どの変数の内容を引き渡すか、その変数を定義する。

➔「変数名」は、1行の範囲内でいくつでも並べて定義することができる。

➔「変数名」に配列変数を指定するときは「(　)(両カッコ)」を付け加えなければならない。

➔すべての変数を引き渡す場合はCHAIN文で「ALL」指定する。COMMON文は、変数のうち一部を引き渡したいときに使用する。

用例

・プログラム1

```
100 CONSOLE 0,25,1,1
110 DIM A(2)
120 COMMON A(),D$
130 INPUT "ｽｳﾁ A(1)=";B
140 INPUT "ｽｳﾁ A(2)=";C
150 A(1)=B
160 A(2)=C
170 INPUT "ｹｲｻﾝ(+ - * /)";D$
180 IF D$="+" THEN 230
190 IF D$="-" THEN 240
200 IF D$="*" THEN 250
210 IF D$="/" THEN 260
220 GOTO 110
230 CHAIN "2:COMMON2.BAS",200
240 CHAIN "2:COMMON2.BAS",210
250 CHAIN "2:COMMON2.BAS",220
260 CHAIN "2:COMMON2.BAS",230
270 END
```

・プログラム2（1から呼び出される。ディスクにセーブしておく）

```
100 PRINT "ｺﾉ ｷﾞｮｳ ﾊ ﾋｮｳｼﾞ ｻﾚﾏｾﾝ"
110 '
200 PRINT A(1)+A(2) :GOTO 240
210 PRINT A(1)-A(2) :GOTO 240
220 PRINT A(1)*A(2) :GOTO 240
230 PRINT A(1)/A(2) :GOTO 240
240 PRINT
250 PRINT "A=";A
260 PRINT "B=";B
270 PRINT "D$= ";D$
280 END
```

参照　CHAIN

OPEN（オープン）

■ファイルを開きファイル操作を開始する

書式　① OPEN "ドライブ名：ファイル名" FOR INPUT AS #ファイル番号

② OPEN "ドライブ名：ファイル名" FOR OUTPUT AS #ファイル番号

③ OPEN "ドライブ名：ファイル名" FOR APPEND AS #ファイル番号

④ OPEN "ドライブ名：ファイル名" #ファイル番号

⑤ OPEN ファイルディスクリプタ #ファイル番号

➔「"ドライブ名：ファイル名"」で指定されたファイルのアクセスを開始するために、「ファイル番号」を割り当てて、ファイルをオープン（開く）する。

➔「ファイル番号」は、BASIC起動時に指定したファイル数1から

13の間の番号を指定し、以降ファイルの指定はこの番号で行なう。

➔書式①は、ディスク上に登録されているシーケンシャルファイルから、データを読み込むことを指定する。

➔書式②は、ディスク上に新しくシーケンシャルファイルを作成し、データを書き込むことを指定する。

➔書式③は、ディスク上に登録されているシーケンシャルファイルの最後へ、データを追加して書き込むことを指定する。

➔書式④は、ランダムアクセスを行なうことを指定する。

➔指定したファイルがディスク上になかった場合、書式①、②、③ではエラーとなる。書式④では新たにファイルが作られる。

➔書式②では、常に新しいファイルを作るために、ディスク上に同一のファイル名が存在した場合、そのファイルは削除されてしまうので注意する。

➔また OPEN 文は、書式⑤のように、ディスクのほかに「ファイルディスクリプタ」で指定して、RS−232C、プリンタ、スクリーンディスプレイ、キーボード、カセットなどを、入出力のためのファイルとしてオープンすることができる。

用例

```
100 PRINT "ｺﾉ ﾌﾟﾛｸﾞﾗﾑ ﾊ OPENﾌﾞﾝ ﾉ"
110 PRINT "ﾂｶｲｶﾀ ｦ ｼﾒｼﾀ ﾓﾉﾃﾞｽ     "
120 PRINT "ｼﾞｯｺｳ ｼﾃﾓ ｴﾗｰ ﾆ ﾅﾘﾏｽ  "
130 OPEN "2:DATA.DAT" FOR INPUT AS #1
140 OPEN "2:DATA.DAT" FOR OUTPUT AS #1
150 OPEN "2:DATA.DAT" FOR APPEND AS #1
160 OPEN "2:DATA.DAT" AS #1
```

参照　CLOSE／PRINT #／INPUT #／FIELD／PUT #／GET #

CLOSE 〔クローズ〕

■ファイルを閉じファイル操作を終了する

書式　① CLOSE
　　　② CLOSE #ファイル番号, #ファイル番号, ………

➔ OPEN 文で開かれたファイルを閉じて、ファイルの操作を終了する。以降、閉じられたファイル番号を、他のファイル操作に割り当てることができる。

➔ 書式①は、現在オープンされているすべてのファイルを閉じる。

➔ 書式②は、「ファイル番号」で指定されたファイルのみを閉じる。「ファイル番号」は「,（カンマ）」で区切って複数指定することができる。

➔ CLOSE 文は、バッファに残っているデータをディスクにはき出す働きをするので、OPEN 文によって開かれたファイルは、この命令によってファイル操作を終了しないと、正常に終了することができない。

➔ END 文または NEW コマンドを実行することによって、ファイルを正常に閉じることができる。STOP 文は、ファイルのクローズを行なわないので、注意する。

➔ RUN または LOAD コマンド（R指定なし）は、プログラムの実行前に開かれているすべてのファイルをクローズする。

用例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATA.DAT" FOR OUTPUT AS #1
120 N=N+1
130 INPUT "NAME =";NA$
140 IF NA$="" THEN 200
150 INPUT "TEL  =";TL$
```

```
160 PRINT #1,N
170 PRINT #1,NA$
180 PRINT #1,TL$
190 GOTO 120
200 CLOSE #1
210 END
```

参照　OPEN／END／NEW／STOP／RUN／LOAD／CHAIN

PRINT #（ナンバー シャープ）

■シーケンシャルファイルにデータを書き込む

書式　①　PRINT　#ファイル番号，データ

②　PRINT　#ファイル番号，データ；データ；………

➔ OPEN 文によって「ファイル番号」でオープンされたファイルに、「データ」を書き込む。

➔「データ」は、数値または文字列を式、変数、配列変数で指定するが、PRINT 文で画面に表示するのと同じようにディスクに出力するので、区切り記号を使って複数のデータを書き込む場合には、データを正しく読み込めるように注意しなければならない。

➔書式①は、1つのデータをディスクに書き込む。

➔書式②は、複数のデータをディスクに書き込む。この場合、ディスプレイに表示されるのと同じように、複数のデータを結合して1つのデータとしてディスクに書き込んでしまうので、注意する。

➔複数のデータをディスクに書き込み、INPUT　# 文で変数に読み込むとき、数値データのデータは適当に区切られて読み込むことができる。しかし、文字列データの場合は、すべてのデータが1つのデータとしてディスクに書き込まれている。そのため、データを

PRINT　#文の変数並びと同一にして、複数のデータとして読み込もうとすると「Input　past　end」エラーが発生して、実行がストップしてしまう場合があるので、注意する。

➔文字列のデータを正しく書き込むには、区切りの記号「,（カンマ)」、「CHR$ (13)（キャリッジリターン)」をデータとして書き込まなければならない。

用　例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATA.DAT" FOR OUTPUT AS #1
120 N=N+1
130 INPUT "NAME =";NA$
140 IF NA$="" THEN 200
150 INPUT "TEL  =";TL$
160 PRINT #1,N
170 PRINT #1,NA$
180 PRINT #1,TL$
190 GOTO 120
200 CLOSE #1
210 END
```

参　照　PRINT／WRITE #／PRINT # USING／INPUT #／LINE INPUT #

● PRINT　#　**応用プログラム**

IDコードに名まえの読みを登録する

```
100 '************************************
110 '               電話帳
120 '************************************
130 CONSOLE 0,25,1,1
140 WIDTH 80,25
150 CLS 3
160 MX=200
170 DIM DT$(MX,5)
180 ON ERROR GOTO *ERRO
```

```
190 OPEN "TELMEM.DAT" FOR INPUT AS #1
200 INPUT #1,NO
210   FOR I=1 TO NO
220     FOR J=1 TO 5
230       INPUT #1,DT$(I,J)
240     NEXT J
250   NEXT I
260 CLOSE
270 ON ERROR GOTO 0
280 '-------------------------------------
290             *MAIN 'ﾒｲﾝ ﾙｰﾁﾝ
300 '-------------------------------------
310 CLS:SL=0:Y=0
320 LOCATE 28,0:PRINT "T E L   M E M O"
330 LOCATE 30,2:PRINT "ﾄｳﾛｸ      :  1"
340 LOCATE 30,3:PRINT "ﾃｲｾｲ      :  2"
350 LOCATE 30,4:PRINT "ｻｸｼﾞｮ     :  3"
360 LOCATE 30,5:PRINT "ﾋｮｳｼﾞ     :  4"
370 LOCATE 30,6:PRINT "ｹﾝｻｸ      :  5"
380 LOCATE 30,7:PRINT "ｷﾛｸ       :  6"
390 LOCATE 30,8:PRINT "ｼｭｳﾘｮｳ    :  7"
400 LOCATE 16,10:INPUT "ﾊﾞﾝｺﾞｳ ﾃﾞ ｴﾗﾝﾃﾞ ｸﾀﾞｻｲ :
 ",SL
410 IF SL<1 OR 7<SL THEN BEEP:GOTO *MAIN
420 CLS
430 LOCATE 1,0 :PRINT"No.  ID              ﾅ ﾏ ｴ"
440 LOCATE 34,0:PRINT" ｼﾞﾀｸ TEL        ｶｲｼｬ TEL
    FAX No."
450 ON SL GOSUB *INPT,*CRCT,*DELT,*DSPL,*SRCH,
*FILE,*QUIT
460 LOCATE 10,22:INPUT "ﾅﾆｶ ｷｰ ｦ ｵｽﾄ ﾒﾆｭｰ ﾆ ﾓﾄﾞ
ﾘﾏｽ",KY$
470 GOTO *MAIN
480 *QUIT
490 CLS:LOCATE 20,7:INPUT "ｼｭｳﾘｮｳ  ( 'Y'es or
N'o )";ANS$
500 IF ANS$<>"Y" AND ANS$<>"y" THEN *MAIN
510 GOSUB *FILE
520 END
530 '
540 *INPT 'ﾃﾞｰﾀ ﾆｭｳﾘｮｸ ------------------
550   NO=NO+1
560   IF MX<NO THEN LOCATE 0,21:PRINT"ｺﾚｲｼﾞｮｳ D
ATA ｦ ﾌﾔｾﾏｾﾝ":RETURN
570   LOCATE 0,0:PRINT "DATA ﾊﾞﾝｺﾞｳ ";NO
580   LOCATE 0,3:INPUT "ID ｺｰﾄﾞ (6 ﾓｼﾞ ﾏﾃﾞ) :",
```

- 190–200: データファイルを開く
- 210–250: ファイルのデータを配列変数に読み込む
- 260: ファイルを閉じる
- 310–400: メニュー表示
- 510–520: プログラム終了時にデータファイルを書き込む

```
DT$(NO,1)
590   LOCATE 0,4:INPUT "ｼﾒｲ       (22ﾓｼﾞ ﾏﾃﾞ) :",
DT$(NO,2)
600   LOCATE 0,5:INPUT "ｼﾞﾀｸ TEL (16ﾓｼﾞ ﾏﾃﾞ) :",
DT$(NO,3)
610   LOCATE 0,6:INPUT "ｶｲｼｬ TEL (16ﾓｼﾞ ﾏﾃﾞ) :",
DT$(NO,4)
620   LOCATE 0,7:INPUT "FAX No.  (16ﾓｼﾞ ﾏﾃﾞ) :",
DT$(NO,5)
630   RETURN
640 *CRCT 'ｼｭｳｾｲ -----------------------
650   LOCATE 0,21:INPUT "ﾅﾝﾊﾞﾝ ﾉ ﾃﾞｰﾀ ｦ ﾃｲｾｲ ｼﾏ
ｽｶ : ";SN
660   IF SN<1 OR NO<SN THEN BEEP:GOTO 690
670   LO=NO:NO=SN-1:CLS:GOSUB *INPT
680   NO=LO
690   RETURN
700 *DELT 'ｻｸｼﾞｮ -----------------------
710   LOCATE 0,21:INPUT "ﾅﾝﾊﾞﾝ ﾉ ﾃﾞｰﾀ ｦ ｻｸｼﾞｮ ｼﾏ
ｽｶ : ";SN
720   IF SN<1 OR NO<SN THEN BEEP:GOTO 830
730   FOR I=SN TO NO
740     FOR J=1 TO 5
750       DT$(I,J)=DT$(I+1,J)
760     NEXT J
770   NEXT I
```

データを削除して後を詰める

```
780   NO=NO-1
790   RETURN
800 *DSPL 'ﾋｮｳｼﾞ -----------------------
810   LOCATE 0,21:PRINT "ﾅﾆｶ ｷｰ ｦ ｵｽﾄ ﾂｷﾞ ﾉ ﾍﾟｰｼﾞ
 ｦ ﾋｮｳｼﾞｼﾏｽ"
820   FOR I=1 TO NO
830     Y=Y+1
840     IF 20<Y THEN 850 ELSE 880
850       ANS$=INKEY$:IF ANS$="" THEN 850
860       Y=1:CLS
870       LOCATE 0,21:PRINT "ﾅﾆｶ ｷｰ ｦ ｵｽﾄ ﾂｷﾞ ﾉ ﾍﾟ
ｰｼﾞ ｦ ﾋｮｳｼﾞｼﾏｽ"
880     DN=I:GOSUB *DPRN
890   NEXT I
900   RETURN
910 *SRCH 'ｹﾝｻｸ -----------------------
920   LOCATE 0,21:INPUT "ｹﾝｻｸｽﾙ ID ｺｰﾄﾞ ﾊ : ";SR$
930   FOR I=1 TO NO
940     IF INSTR(DT$(I,1),SR$)=0 THEN 1010
```

文字列を探す

```
950         Y=Y+1
960         IF 20<Y THEN 970 ELSE 1000
970           ANS$=INKEY$:IF ANS$="" THEN 970
980           Y=1:CLS
990           LOCATE 0,21:PRINT "ﾅﾆｶ ｷｰ ｦ ｵｽﾄ ﾂｷﾞ ﾉ
 ﾍﾟｰｼﾞ ｦ ﾋｮｳｼﾞｼﾏｽ"
1000          DN=I:GOSUB *DPRN
1010        NEXT I
1020      RETURN
1030 *FILE 'ｷﾛｸ ---------------------------
1040      LOCATE 15,10:INPUT "ﾌﾛｯﾋﾟｰ ﾉ ｼﾞｭﾝﾋﾞ OK ( 'Y
'es or 'N' )";KY$
1050      IF KY$="Y" THEN 1060 ELSE 1140
1060        OPEN "TELMEM.DAT" FOR OUTPUT AS #1
1070        PRINT #1,NO
1080        FOR I=1 TO NO
1090          FOR J=1 TO 5
1100            PRINT #1,DT$(I,J)
1110          NEXT J
1120        NEXT I
1130      CLOSE
1140      RETURN
1150 *DPRN 'ﾃﾞｰﾀ ﾋｮｳｼﾞ -------------------
1160      LOCATE 0,Y:PRINT DN;
1170      LOCATE 4,Y:PRINT DT$(DN,1);
1180      LOCATE 10,Y:PRINT DT$(DN,2);
1190      LOCATE 32,Y:PRINT DT$(DN,3);
1200      LOCATE 48,Y:PRINT DT$(DN,4);
1210      LOCATE 64,Y:PRINT DT$(DN,5);
1220      RETURN
1230 *ERRO 'ｴﾗｰ ｼｮﾘ ----------------------
1240      IF ERL=190 THEN RESUME 260
1250      ON ERROR GOTO 0
```

●「電話帳」プログラムの使い方

RUN すると、処理選択のメニュー画面になる。データの最大件数は、160行の変数MXに設定されている200件である。メニュー番号を入力して処理を選ぶ。

IDコードには、アルファベットやカナ文字での名まえの読みを登録するとよい。検索は、このIDコードをもとに行なう。氏名には、漢字を使うこともできる。

WRITE　#（ライト　シャープ）

■シーケンシャルファイルにデータを書き込む

書式　① WRITE　#ファイル番号，データ

② WRITE　#ファイル番号，データ，データ，………

➔ OPEN 文によって「ファイル番号」でオープンされたファイルに、「データ」を書き込む。

➔「データ」は、数値または文字列を式、変数、配列変数で指定するが、WRITE 文で画面に表示するのと同じようにディスクに出力するので、区切り記号を使って複数のデータを書き込む場合には、不要な空白を詰めて、データの間を必ず「,（カンマ）」で区切って書き込まれる。

また、文字列データは「"（ダブルクォーテーションマーク）」で囲まれて書き込まれる。

➔ 書式①は、1つのデータをディスクに書き込む。

➔ 書式②は、複数のデータをディスクに書き込む。この場合、PRINT　#文と違って、ひとつひとつのデータを「,（カンマ）」で区切って書き込むので、INPUT　#文で変数に読み込む際に、変数のデータ並びに応じてひとつひとつ読み込むことができる。

➔ すべてのデータを書き込むと、改行コードを書き込む。データを省略した場合は、単に改行コードのみを書き込む。

用例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATA.DAT" FOR OUTPUT AS #1
120 N=N+1
130 INPUT "NAME =";NA$
140 IF NA$="" THEN 180
150 INPUT "TEL  =";TL$
160 WRITE #1,N,NA$,TL$
```

```
170 GO[redacted]
180 CL[redacted]
190 EN[redacted]
```

参照　WRITE／PRINT ＃／INPUT ＃／LINE INPUT ＃

PRINT # USING
(プリント　シャープ　ユージング)

■文字列、数値などを編集してファイルに出力する

書式　PRINT #ファイル番号，USING "書式制御文字列"；
データ；データ………

➔「ファイル番号」で指定されたファイルへ、「データ」を「書式制御文字列」で指定した書式に編集して書き込む。

➔日本語を含む文字列は編集することはできない。

➔「書式制御文字列」は、指定した数値の出力範囲を超えた数値を出力する場合、または四捨五入などで丸められた数値が出力範囲を超えた場合などは、数値の先頭に「％(パーセント)」が付加されて注意を促す。

➔PRINT ＃ USING文は、ファイルに出力する以外は、PRINT USING文と同様である。

➔おもな「書式制御文字列」には以下のようなものがある。

＃(ナンバー)………出力する数値の桁数を指定して、右詰めで編集する。

＆　　＆(アンパーサンドにはさまれたスペース)
………文字列や文字変数の内容から、スペースの数＋アンパーサンドの数分の文字列を先頭から切りとって、左詰めで編集する。

用　例

```
100 CONSOLE 0,25,1,1
110 OPEN "B:DATA.DAT" FOR OUTPUT AS #1
120 N=N+1
130 INPUT "NAME =";NA$
140 IF NA$="" THEN 260
150 INPUT "KOKUGO";KO
160 INPUT "SANSU ";SA
170 INPUT "RIKA  ";RI
180 INPUT "SHAKAI";SH
190 PRINT #1,N
200 PRINT #1,USING"&                    &";NA$
210 PRINT #1,USING"########";KO
220 PRINT #1,USING"########";SA
230 PRINT #1,USING"########";RI
240 PRINT #1,USING"########";SH
250 GOTO 120
260 CLOSE #1
270 END
```

参　照　PRINT USING／PRINT ＃／OPEN

INPUT #（インプット シャープ）

シーケンシャルファイルからデータを読み込む

書 式　① INPUT #ファイル番号，変数

② INPUT #ファイル番号，変数，変数，………

➔ OPEN 文によって「ファイル番号」でオープンされたシーケンシャルファイルから、「変数」にデータを読み込む。

➔シーケンシャルファイルからデータを読み込むほかは、INPUT 文と同様の働きをする。

➔読み込まれるデータの型と、読み込む「変数」の型は同じでなければならない。

➔数値変数には「 （スペース）」、「，（カンマ）」、「CHR$ (13) （キャリッジリターン）」で区切られた数字が読み込まれる。

➔文字列変数には、「，（カンマ）」、「CHR$(13) （キャリッジリターン）」で区切られた文字列が読み込まれる。

用 例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATA.DAT" FOR INPUT AS #1
120 IF EOF(1)=-1 THEN 190
130   INPUT #1,N,NA$,TL$
140   PRINT:PRINT N
150   PRINT "NAME =";NA$
160   PRINT "TEL  =";TL$
170 GOTO 120
180 CLOSE #1
190 END
```

参 照　OPEN／PRINT #／WRITE #／LINE INPUT #

LINE INPUT
(ライン　インプット　シャープ)

1行単位でデータを読み込む

書式　LINE　INPUT　#ファイル番号, 文字変数

➔ OPEN文によって「ファイル番号」でオープンされたシーケンシャルファイルから、「文字変数」にデータを1行単位で読み込む。

➔ 1行単位とは、シーケンシャルファイルのデータの最初から、改行コード「CHR$(13)（キャリッジリターン）」までのすべての文字で、LINE　INPUT　#文は、このコードまでのデータを一括して文字変数に読み込む。

➔ LINE　INPUT　#文が続くと、次のLINE　INPUT　#文では、改行コードとラインフィードコード「CHR$(10)(ラインフィード)」の組みを読み飛ばして、次の改行コードまでの文字を読み込む。

➔ LINE　INPUT　#文によって、データファイルやアスキーセーブでディスクに登録されているプログラムを、他のプログラムから直接データとして読み込むことができる。

用例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATA.DAT" FOR INPUT AS #1
120 IF EOF(1)=-1 THEN 170
130   LINE INPUT #1,DT$
140   PRINT DT$
150 GOTO 120
160 CLOSE #1
170 END
```

参照　OPEN／INPUT　#

FIELD（フィールド）

ランダムファイルのフォーマットを設定する

書式　FIELD　#ファイル番号，フィールド長　AS　文字変数，フィールド長　AS　文字変数，………

➔ OPEN 文によって「ファイル番号」でオープンされたランダムファイルに1レコードのデータを書き込む、あるいは読み出す準備をする。このために、ランダムアクセスバッファに「フィールド長」と「文字変数」で、書き込み、読み出しの領域を割り当て、レコードのフォーマットを設定する。

➔「フィールド長」とは、「文字変数」に代入するデータの文字数で、1レコードあたり合計256文字までの範囲で、複数の「文字変数」を割り当てることができる。

➔「文字変数」は、ランダムアクセス専用とし、プログラム中で他の入力（INPUT 文など）や通常の代入式の左辺の変数として使用しないこと。ランダムアクセス以外に使用した場合、一般の変数として扱われ、それ以降ランダムファイルの入出力が正常に行なわれなくなるので、注意する。

➔ FIELD 文は、必ず PUT　#文、GET　#文の使用に先だって実行されていなければならない。

➔同じ「ファイル番号」を使って、異なった形式の FIELD 文を実行することができる。この場合、同じ変数名があとから指定されると、それまでの割り当てが無効になる。一般に、多数の変数を1フィールドに割り当てる場合に、プログラムを見やすくするのに使われる。

用例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATARD.DAT" AS #1
```

```
120 FIELD #1,4 AS NOF$,20 AS NAF$,20 AS TLF$
130 N=N+1
140 INPUT "NAME  =";NA$
150 IF NA$="" THEN 220
160 INPUT "TEL   =";TL$
170 LSET NOF$=MKI$(N)
180 LSET NAF$=NA$
190 LSET TLF$=TL$
200 PUT #1
210 GOTO 130
220 CLOSE #1
230 END
```

参照　OPEN／GET #／PUT #／LSET／RSET／MKI$／MKS$／MKD$／CVI／CVS／CVD

●FIELD　応用プログラム

ソートルーチンを加えればより実用的に

```
100 '************************************
110 '        データベース カード
120 '************************************
130 CONSOLE 0,25,1,1
140 WIDTH 80,25
150 CLS 3
160 MX=500 ―――――― カードの最大枚数
180 OPEN "DBCARD.DAT" AS #1 ―┐
190 FIELD #1,255 AS DF$ ―――┘― ランダムファイルのオープン
200 ON ERROR GOTO *ERRO
210 OPEN "DBCARD.IDX" FOR INPUT AS #2 ― シーケンシャルファイルのオープン
220 INPUT #2,MN ―― カードの枚数の読み込み
230 CLOSE #2
240 ON ERROR GOTO 0
250 '------------------------------------
260             *MAIN 'メイン ルーチン
270 '------------------------------------
280 CLS:SL=0
```

```
290 LOCATE 28,0:PRINT "DATA BASE CARD"
300 LOCATE 30,2:PRINT "ﾂｲｶ      :  1"
310 LOCATE 30,3:PRINT "ﾃｲｾｲ     :  2"
320 LOCATE 30,4:PRINT "ｻｸｼﾞｮ    :  3"
330 LOCATE 30,5:PRINT "ﾋｮｳｼﾞ    :  4"
340 LOCATE 30,6:PRINT "ｲﾁﾗﾝ     :  5"
350 LOCATE 30,7:PRINT "ｹﾝｻｸ     :  6"
360 LOCATE 30,8:PRINT "ｼｭｳﾘｮｳ   :  7"
370 LOCATE 16,10:INPUT "ﾊﾞﾝｺﾞｳ ﾃﾞ ｴﾗﾝﾃﾞ ｸﾀﾞｻｲ :
 ",SL
380 IF SL<1 OR 7<SL THEN BEEP:GOTO *MAIN
390 CLS
400 LOCATE 0,0 :PRINT"CARD No.";NO;"   MAX CARD
 No.";MN
410 ON SL GOSUB *INPT,*CRCT,*DELT,*DSPL,*DLST,*
SRCH,*QUIT
420 LOCATE 0,22:INPUT "ﾅﾆｶ ｷｰ ｦ ｵｽﾄ ﾒﾆｭｰ ﾆ ﾓﾄﾞﾘ
ﾏｽ    ",KY$
430 GOTO *MAIN
440 *QUIT
450 CLS:LOCATE 20,7:INPUT "ｼｭｳﾘｮｳ  ( 'Y'es or '
N'o )";KY$
460 IF KY$<>"Y" AND KY$<>"y" THEN *MAIN
470 OPEN "DBCARD.IDX" FOR OUTPUT AS #2
480 PRINT #2,MN
490 CLOSE
500 END
510 '
520 *INPT 'ﾃﾞｰﾀ ﾂｲｶ ----------------------
530   MN=MN+1
540   IF MX<MN THEN LOCATE 0,21:PRINT"ｺﾚｲｼﾞｮｳ C
ARD ｦ ﾌﾔｾﾏｾﾝ":RETURN
550   LOCATE 0,0:PRINT "CARD No.";MN;"    Max C
ARD No.";MN
560   LOCATE 0,2:INPUT "CARD DATA (255 ﾓｼﾞ ﾏﾃﾞ)
 :",DT$
570   LSET DF$=DT$
580   PUT #1,MN
590   RETURN
600 *CRCT 'ｼｭｳｾｲ ------------------------
610   LOCATE 0,21:INPUT "ﾅﾝﾊﾞﾝ ﾉ ﾃﾞｰﾀ ｦ ﾃｲｾｲ ｼﾏ
ｽｶ : ";SN
620   IF SN<1 OR MN<SN THEN BEEP:GOTO 690
630   LOCATE 0,0:PRINT "CARD No.";SN;"    Max C
ARD No.";MN
```

290～370行 メニュー表示

470～490行 プログラム終了時にカードの枚数をシーケンシャルファイルに書き込み、すべてのファイルをクローズする

570～580行 カードのデータをランダムファイルへ書き込む

```
640    GET #1,SN ─────┐
650    DT$=DF$ ───────┘── 修正させるデータを読み込む
660    LOCATE 0,2:PRINT "CARD DATA (255 ﾓｼﾞ ﾏﾃﾞ)
 :";DT$
670    LO=MN:MN=SN-1:GOSUB *INPT
680    MN=LO
690    RETURN
700  *DELT 'ｻｸｼﾞｮ -------------------------
710    LOCATE 0,21:INPUT "ﾅﾝﾊﾞﾝ ﾉ CARD ｦ ｻｸｼﾞｮ ｼ
 ﾏｽｶ : ";SN
720    IF SN<1 OR MN<SN THEN BEEP:GOTO 960
730    LOCATE 0,0:PRINT "CARD No.";SN;"     Max C
 ARD No.";MN
740    GET #1,SN ─────┐
750    DT$=DF$ ───────┘── 削除するデータを確認のために読み込む
760    LOCATE 0,2:PRINT DT$
770    LOCATE 0,21:INPUT "ｺﾉ CARD ｦ ｻｸｼﾞｮ ｼﾏｽ (
 Y or N ) :";KY$
780    IF KY$="Y" OR KY$="y" THEN 790 ELSE 820
790    DT$="" ────────┐
800    LSET DF$=DT$   ├── カラのデータを書き込んで削除する
810    PUT #1,SN ─────┘
820    RETURN
830  *DSPL 'ﾋｮｳｼﾞ -------------------------
840    LOCATE 0,21:INPUT "ﾅﾝﾊﾞﾝ ﾉ CARD ｦ ﾋｮｳｼﾞ ｼ
 ﾏｽｶ : ",SN
850    IF SN<1 OR MN<SN THEN BEEP:GOTO 840
860    LOCATE 0,0:PRINT "CARD No.";SN;"     Max C
 ARD No.";MN
870    GET #1,SN ─────┐
880    DT$=DF$ ───────┘── 表示するデータを読み込む
890    LOCATE 0,2:PRINT DT$
900    RETURN
910  *DLST 'ｲﾁﾗﾝ -------------------------
920    FOR I=1 TO MN
930      CLS
940    LOCATE 0,21:PRINT "ﾅﾆｶ ｷｰ ｦ ｵｽﾄ ﾂｷﾞ ﾉ CARD
 ｦ ﾋｮｳｼﾞｼﾏｽ"
950    LOCATE 0,22:PRINT "'E' ｷｰ ｦ ｵｽﾄ ﾋｮｳｼﾞ ｦ ｼ
 ｭｳﾘｮｳ ｼﾏｽ"
960      LOCATE 0,0:PRINT "CARD No.";I;"     Max
 CARD No.";MN
970      GET #1,I ─────┐   920～1020行のFOR～NEXT文で
980      DT$=DF$ ──────┘── つぎつぎにデータを読み込んで表示する
990      LOCATE 0,2:PRINT DT$
```

```
1000       KY$=INKEY$:IF KY$="" THEN 1000
1010       IF KY$="E" THEN 1030 ELSE 1020
1020     NEXT I
1030     RETURN
1040   *SRCH 'ケンサク ------------------------
1050     LOCATE 0,21:INPUT "ケンサクスル モジ゛ ハ : ";SR$
1060     FOR I=1 TO MN
1070       CLS
1080       LOCATE 0,21:PRINT "ケンサクスル モジ゛ ハ : ";SR
 $
1090       LOCATE 0,22:PRINT "'E' キー ヲ オスト ヒョウジ゛
 ヲ シュウリョウ シマス"
1100       GET #1,I        1100～1120行カードのデータを読み込んで
1110       DT$=DF$                    検索文字列と比較する
1120       IF INSTR(DT$,SR$)<>0 THEN 1130 ELSE 1170
1130         LOCATE 0,0:PRINT "CARD No.";I;"     M
ax CARD No.";MN
1140         LOCATE 0,2:PRINT DT$
1150         KY$=INKEY$:IF KY$="" THEN 1150
1160         IF ANS$<>"E" THEN 1170 ELSE 1180
1170     NEXT I
1180     RETURN
1190   *ERRO 'エラー ショリ --------------------
1200     IF ERL=210 THEN RESUME 240
1210     ON ERROR GOTO 0
```

●「データベースカード」プログラムの使い方

RUNすると、処理選択のメニュー画面になる。データの最大件数は、160行の変数「MX」に設定されている500件である。メニュー番号を入力して処理を選ぶ。

データには、漢字を使うこともできる。訂正や削除などの処理は、カードの番号を指定して行なう。目次となるインデックスファイルを使うようにして、ソート（並べ替え）などのルーチンを加えるとより実用的になる。

LSET／RSET
(レフトセット／ライトセット)

ランダムファイルバッファにデータを設定する

書式　① LSET　文字変数＝文字列データ
　　　② RSET　文字変数＝文字列データ

➔ データをディスク上のランダムファイルに書き込むために、FIELD 文でランダムアクセスバッファに割り当てられた「文字変数」に「文字列データ」を設定する。

➔ 書式①は、「文字列データ」を左詰めに「文字変数」に設定する。文字列データが、FIELD 文で設定されたフィールド長より長い場合は、文字列右側の超過部分が切り捨てられる。

➔ 書式②は、「文字列データ」を右詰めに「文字変数」に設定する。文字列データが、FIELD 文で設定されたフィールド長より長い場合は、文字列右側の超過部分が切り捨てられる。

➔ 設定するデータは、必ず文字型のデータでなければならない。そのため、数値型データは、MKI$、MKS$、MKD$で文字型データに変換しておく必要がある。

➔ データは、LSET／RSET 文で変数にセットされたデータから、FIELD 文で設定したランダムアクセスバッファを介して、ディスク上のランダムファイルに書き込まれる。

用例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATARD.DAT" AS #1
120 FIELD #1,4 AS NOF$,20 AS NAF$,20 AS TLF$
130 N=N+1
140 INPUT "NAME  =";NA$
150 IF NA$="" THEN 220
```

```
160 INPUT "TEL    =";TL$
170 RSET NOF$=MKI$(N)
180 LSET NAF$=NA$
190 LSET TLF$=TL$
200 PUT #1
210 GOTO 130
220 CLOSE #1
230 END
```

参照　FIELD／PUT #／GET ／MKI$／MKS$／MKD$

PUT #（プット シャープ）

■ランダムファイルにデータを書き込む

書式　PUT #ファイル番号，レコード番号

➔ OPEN 文によって「ファイル番号」でオープンされたランダムファイルに、FILED 文で設定されたランダムアクセスバッファ内のデータを書き込む。

➔「レコード番号」は、ランダムファイルのレコード（1～65000）のうちから、書き込むレコードを指定する。

➔「レコード番号」を省略すると、直前に行なわれたPUT #、GET #文で参照されたレコードの次のレコードが指定されたとみなされる。

➔書き込むデータは、あらかじめ FIELD 文、LSET、RSET 文でランダムアクセスバッファ内のフィールドにセットされていなければならない。

➔ PUT #文は、プリンタやディスプレイにバッファ内のデータを出力することもできる。この場合、事前にプリンタ、またはディスプレイをファイルとしてオープンしておく必要がある。

用例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATARD.DAT" AS #1
120 FIELD #1,4 AS NOF$,20 AS NAF$,20 AS TLF$
130 N=N+1
140 INPUT "NAME  =";NA$
150 IF NA$="" THEN 220
160 INPUT "TEL   =";TL$
170 RSET NOF$=MKI$(N)
180 LSET NAF$=NA$
190 LSET TLF$=TL$
200 PUT #1
210 GOTO 130
220 CLOSE #1
230 END
```

参照　OPEN／FIELD／LSET／RSET／GET ＃

GET #（ゲット　シャープ）

■ランダムファイルからデータを読み込む

書式　GET #ファイル番号，レコード番号

➔ OPEN 文によって「ファイル番号」でオープンされたランダムファイルから、FILED 文で設定されたランダムアクセスバッファへ、「レコード番号」で指定したデータを読み込む。

➔「レコード番号」は、ランダムファイルのレコード（1～65000）のうちから、読み込むデータのあるレコードを指定する。

➔「レコード番号」を省略すると、直前に行なわれたPUT ＃、GET ＃文で参照されたレコードの次のレコードが指定されたとみなされる。

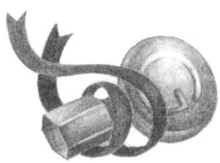
➔読み込むデータは、あらかじめ FIELD 文でランダムアクセスバッファ内にフィールドを設定して、準備されていなければならない。

➔ GET #文は、キーボードやカセットからバッファ内にデータを取り込むこともできる。この場合、事前にキーボードやカセットをファイルとしてオープンしておく必要がある。

用　例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATARD.DAT" AS #1
120 FIELD #1,4 AS NOF$,20 AS NAF$,20 AS TLF$
130 PRINT "ﾚｺｰﾄﾞ 1 -";LOF(1)
140 INPUT "ﾅﾝﾊﾞﾝ ﾉ ﾚｺｰﾄﾞ ｦ ﾖﾐｺﾐ ﾏｽｶ ";RC
150 IF RC<1 OR LOF(1)<RC THEN BEEP:GOTO 220
160 GET #1,RC
170 PRINT
180 PRINT CVI(NOF$)
190 PRINT NAF$
200 PRINT TLF$
210 GOTO 130
220 CLOSE #1
230 END
```

参　照　　OPEN／FIELD／LSET／RSET／PUT #



MKI$（メイクインテジャーダラー）

MKS$（メイクシングルダラー）

MKD$（メイクダブルダラー）

■数値データを内部表現に対応する文字列に変換

書式　① MKI$　（整数型数値データ）

② MKS$　（単精度型数値データ）

③ MKD$　（倍精度型数値データ）

➔数値データをランダムファイルに記録するために、LSET、RSET文を使用する前に、文字列データに変換するのに用いる関数。

➔書式①は、「整数型数値データ」（−32768から32767までの整数型の数値）を2文字（2バイト）の文字列に変換する。

➔書式②は、「単精度型数値データ」（−1.70141E＋38から1.70141E＋38までの単精度実数型の数値）を4文字（4バイト）の文字列に変換する。

➔書式③は、「倍精度型数値データ」（有効桁16桁の倍精度実数型の数値）を8文字（8バイト）の文字列に変換する。

➔文字列に変換されたデータは、CVI、CVS、CVD 関数によって元の数値に変換することができる。

用例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATAR1.DAT" AS #1
120 FIELD #1,20 AS MOF$,2 AS SEF$,4 AS TAF$,8 AS
 BAF$
130 '-------------------------------------
140 N=N+1
```

```
150 PRINT N
160 INPUT "モジレツ             =";MO$
170 IF MO$="" THEN 280
180 INPUT "セイスウ             =";SE%
190 INPUT "タンセイド゛ジ゛ッスウ  =";TA!
200 INPUT "バ゛イセイド゛ジ゛ッスウ =";BA#
210 LSET MOF$=MO$
220 RSET SEF$=MKI$(SE%)
230 RSET TAF$=MKS$(TA!)
240 RSET BAF$=MKD$(BA#)
250 PUT #1
260 GOTO 140
270 '------------------------------------
280 PRINT:PRINT
290 INPUT "ヨミコム DATA ノ バ゛ンゴ゛ウ ハ";RC
300 IF RC<1 OR LOF(1)<RC THEN 370
310 GET #1,RC
320 PRINT MOF$
330 PRINT CVI(SEF$)
340 PRINT CVS(TAF$)
350 PRINT CVD(BAF$)
360 GOTO 290
370 CLOSE #1
380 END
```

参 照　CVI／CVS／CVD

CVI（コンバートインテジャー）

CVS（コンバートシングル）

CVD（コンバートダブル）

■内部表現に対応した文字列を数値データに変換

書式　① CVI （2文字の文字列）
② CVS （4文字の文字列）
③ CVD （8文字の文字列）

➔ディスク上のランダムファイルに記録されたデータは文字列型のため、GET #文でランダムアクセスバッファに読み込まれた文字列データを、数値型に変換するのに使用する。

➔書式①は、2文字（2バイト）の文字列を「整数型数値データ」（−32768から32767までの整数型の数値）に変換する。

➔書式②は、4文字（4バイト）の文字列を「単精度型数値データ」（−1.70141E+38から1.70141E+38までの単精度実数型の数値）に変換する。

➔書式③は、8文字（8バイト）の文字列を「倍精度型数値データ」（有効桁16桁の倍精度実数型の数値）に変換する。

➔ CVI、CVS、CVD 関数は、文字列のキャラクタコードを浮動小数点形式の数値データとみなして、文字列を数値に変換している。

用例　前ページ「MKI$／MKS$／MKD$」のプログラムを参照。

参照　MKI$／MKS$／MKD$

EOF（エンド　オブ　ア　ファイル）

■ファイルエンドかどうかのコードを返す

書式　EOF（ファイル番号）

➔ OPEN 文によってオープンされている「ファイル番号」のファイルが、ファイルエンド（ファイルの最後の位置）に達しているかどうかを調べる関数。

➔ファイルエンドであれば、−1（真）を返す。

➔ファイルエンドでなければ、0（偽）を返す。

➔一般的にファイルが最後まで読み込まれたかどうかを調べるのに使用する。

➔「ファイル番号」でオープンされているファイルが、RS−232C の場合、EOF関数は、バッファが空（カラ）の状態かどうかを調べ、空（カラ）なら−1（真）を、空（カラ）でなければ0（偽）を返す。

用例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATA.DAT" AS #1
120 INPUT #1,N
130 '----------------------------------
140 I=I+1
150 INPUT #1,DT$(I)
160 PRINT DT$(I)
170 IF EOF(1)=0 THEN 140
180 CLOSE
190 END
```

参照　OPEN／INPUT ＃／LINE INPUT ＃

FPOS（ファイルポジション）

■ディスク上の物理的な現在位置を返す

書式　FPOS（ファイル番号）

➔ OPEN 文によってオープンされている「ファイル番号」のファイルで、この関数が実行される直前に読み込み、または書き込みを行なったディスク上の物理的位置を返す関数。

➔ 返す値は、セクタ番号で返され、この値はヘッド0、トラック0、セクタ1を0番とした通し番号で与えられる。

用例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATA.DAT" AS #1
120 INPUT #1,N
130 '-----------------------------------
140 I=I+1
150 INPUT #1,DT$(I)
160 PRINT FPOS(1)
170 IF EOF(1)=0 THEN 140
180 CLOSE
190 END
```

参照　OPEN

LOC（ロケーションカウンター）

■ディスク上の論理的な現在位置を返す

書式　LOC（ファイル番号）

➔ OPEN 文によってオープンされている「ファイル番号」のファイルで、この関数が実行される直前に読み込み、または書き込みを行なったディスク上の論理的位置を返す関数。

➔「ファイル番号」で指定されたファイルが、ランダムアクセスファイルの場合、最後にPUT　#、GET　#文で書き込み、または、読み込みのあったレコードのレコード番号が返される。

➔「ファイル番号」で指定されたファイルが、シーケンシャルファイルの場合は、そのファイルがオープンされてから最後に、PRINT　#文やINPUT　#文などによって書き込み、または読み込みのあったレコードの総数を返す。

➔「ファイル番号」で指定されたファイルが、RS−232Cの場合は、割り込みによって受け付けられ、入力バッファ内にたまっている文字数を返す。

➔「ファイル番号」で指定されたファイルがキーボードの場合、キーボードからの割り込みによって受け付けられ、キーボードバッファにたまっている文字数を返す。

用　例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATAR1.DAT" AS #1
120 FIELD #1,20 AS MOF$,2 AS SEF$,4 AS TAF$,8 AS
 BAF$
130 INPUT "ﾖﾐｺﾑ DATA ﾉ ﾊﾞﾝｺﾞｳ ﾊ";RC
140 IF RC<1 OR LOF(1)<RC THEN 220
150 GET #1,RC
160 PRINT LOC(1)
170 PRINT MOF$
180 PRINT CVI(SEF$)
190 PRINT CVS(TAF$)
200 PRINT CVD(BAF$)
210 GOTO 130
220 CLOSE #1
230 END
```

参　照　　OPEN

LOF（レングス　オブ　ア　ファイル）

■ディスク上のファイルの大きさを返す

書式　LOF（ファイル番号）

➔ OPEN 文によってオープンされている「ファイル番号」のファイルの大きさをセクタ数で返す関数。

➔「ファイル番号」で指定されたファイルが、ランダムアクセスファイルの場合、そのファイルの最大レコード番号が返される。

➔「ファイル番号」で指定されたファイルが、シーケンシャルファイルの場合、そのファイルの大きさをセクタ数で返す。

➔「ファイル番号」で指定されたファイルが、RS−232C の場合は、バッファの残りバイト数を返す。

用例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATAR1.DAT" AS #1
120 FIELD #1,20 AS MOF$,2 AS SEF$,4 AS TAF$,8 AS
 BAF$
130 PRINT "1 -";LOF(1)
140 PRINT "  ﾉ ﾚｺｰﾄﾞ ﾊﾞﾝｺﾞｳ ﾃﾞ ｼﾃｲｼﾃｸﾀﾞｻｲ"
150 INPUT "ﾖﾐｺﾑ DATA ﾉ ﾊﾞﾝｺﾞｳ ﾊ";RC
160 IF RC<1 OR LOF(1)<RC THEN 230
170 GET #1,RC
180 PRINT MOF$
190 PRINT CVI(SEF$)
200 PRINT CVS(TAF$)
210 PRINT CVD(BAF$)
220 GOTO 150
230 CLOSE #1
240 END
```

参照　OPEN

ATTRB$（アトリビュートダラー）

指定したディスクおよびファイルの属性を返す

書式　① ATTRB$（ドライブ名）
② ATTRB$（#ファイル番号）
③ ATTRB$（“ファイル名”）

➔書式①は、「ドライブ名」で指定した場合のディスクドライブに設定されている属性を返す。書式②は、OPEN文によって「ファイル番号」でオープンされているファイルに設定されている属性を返す。書式③は、「ファイル名」で指定したファイルに設定されている属性を返す。

➔それぞれ返される「属性文字」とその意味は、次のとおりである。

”□□□”………属性なし。属性が解除されていて、読み込み、書き込みともに可能な状態を示す。

”R□□”………リードアフターライト属性。書き込みの際に正確に書き込まれたかどうか、書き込んだ内容とメモリ上の内容を比較してチェックする指定。

”□E□”………Pオプション属性。プログラムが暗号化されて登録されている状態を示す。

”□□P”………プロテクト属性。書き込み禁止の状態を示す。

用例

```
100 CONSOLE 0,25,1,1
110 OPEN "2:DATA.DAT" AS #1
120 PRINT ATTRS(2)
130 PRINT ATTRS(#1)
140 PRINT ATTRS("2:DATA.DAT")
150 CLOSE #1
160 END
```

参照　SET／SAVE

8. 例外処理命令

ON ERROR GOTO ～ RESUME
(オン エラー ゴーツー ～ リジューム)

エラー処理の開始行を設定する

書式

① ON ERROR GOTO エラー処理開始行
② RESUME
③ RESUME NEXT
④ RESUME 再開始行

→ふつうエラーが起こると、エラーメッセージを表示してコマンド待ちの状態に戻るが、エラートラップ（わな）をかけると、エラーが発生した場合に設定されたエラー処理ルーチンへ制御を移し、エラー回復処理を行なった後、RESUME 文によって指定された行に処理を戻すことができる。

→エラーが発生して、エラー処理ルーチンへ実行が移った後、さらにエラー処理ルーチン内でエラーが発生すると、エラーメッセージを表示してプログラムの実行を停止して、コマンド待ちに戻ってしまうので注意する。

→「エラー処理開始行」は、行番号またはラベルで指定することができる。

→エラートラップを解除するには、「エラー処理開始行」に0（ゼロ）を設定して実行する。

→エラー回復処理が終わると、RESUME 文でプログラムの実行を再開する。

➔書式②は、エラーの原因となった文から実行を再開する。

➔書式③は、エラーの原因となった文のすぐ次の文から実行を再開する。

➔書式④は、「再開始行」で指定した行から実行が再開される。「再開始行」は、行番号またはラベルで指定することができる。

用 例

```
100 CONSOLE 0,25,1,1
110 ON ERROR GOTO *EROP
120 OPEN "2:DATA.DAT" FOR INPUT AS #1
130 INPUT #1,N
140 FOR I=0 TO N
150   INPUT #1,DT$
160   PRINT DT$
170 NEXT I
180 CLOSE
190 ON ERROR GOTO 0
200 END
210 '-----------------------------------
220 *EROP
230  IF ERL=120 THEN 240 ELSE 280
240    PRINT "DATA ﾌｧｲﾙ ｶﾞ ｱﾘﾏｾﾝ. ｱﾀﾗｼｸ ﾂｸﾘﾏｽ"
250    OPEN "2:DATA.DAT" FOR OUTPUT AS #1
260    PRINT #1,0
270    PRINT #1,"ｼﾝｷ DATA No.0"
280  CLOSE
290  RESUME 120
```

参 照　ERROR／ERR／ERL

ON　KEY　GOSUB ～ RETURN
(オン　キー　ゴーサブ ～ リターン)

■ f キーによる割り込み処理の開始行を設定する

書式　① ON　KEY　GOSUB　処理開始行, 処理開始行,
　　…………
② RETURN
③ RETURN　戻り行

➔ f・1からf・10までのファンクションキーを押したときに割り込み処理を行なう処理ルーチンの開始行を、それぞれのキー番号に対応して設定する。この命令は、割り込み処理を設定するだけで、処理の制御はKEY　ON／OFF／STOPで行ない、実際の割り込み処理はファンクションキーを押したときに行なわれる。

➔「処理開始行」は、行番号またはラベルで指定する。

➔「処理開始行」の並びは、ファンクションキーの番号に対応している。つまり、一番目はf・1、二番目はf・2という並びで設定する。たとえば、二番目を飛ばして一番目と三番目を設定するには、「100，，300」などと「，(カンマ)」のみを付す。

➔割り込み処理はサブルーチンとして実行されるので、処理ルーチンからの復帰は、RETURN 文で行なう。

➔書式②は、中断したところから再開する。書式③は、行番号またはラベルで指定された「戻り行」から再開する。

用例

```
100 CONSOLE 0,25,1,1
110 ON KEY GOSUB *F1,,*F3,270,310
120 KEY ON
130 PRINT "ファンクションキー 1 カラ 5 デ゛ エランデ゛ クダ゛サイ"
140 PRINT "シュウリョウ ハ スペ゜ースハ゛ー ヲ オシテ クダ゛サイ"
```

```
150 KY$=INKEY$:IF KY$<>" " THEN 150
160 KEY OFF
170 END
180 '-----------------------------------
190 *F1
200  PRINT "*F1 ニ キマシタ"
210  PRINT
220  RETURN
230 *F3
240  PRINT "*F3 ニ キマシタ"
250  PRINT
260  RETURN
270 ' F4
280  PRINT "270 ギョウ ニ キマシタ"
290  PRINT
300  RETURN
310 ' F5
320  PRINT "310 ギョウ ニ キマシタ"
330  PRINT
340  RETURN
```

参照　KEY　ON／OFF／STOP

KEY　ON／OFF／STOP
(キー　オン／オフ／ストップ)

■fキーによる割り込みを制御する

書式
① KEY　(キー番号)　ON
② KEY　(キー番号)　OFF
③ KEY　(キー番号)　STOP

➔ファンクションキーが押された場合に、割り込みの許可、禁止、停止を制御する。

➔書式①は、割り込みを許可し、ファンクションキーが押されるたびに、ON　KEY　GOSUB文で設定されたキーに対応する処理ルーチンに実行を移す。

➔書式②は、割り込みを禁止し、以降、ファンクションキーが押されても処理ルーチンへは実行を移さず、通常のファンクションキーの動作が行なわれる。

➔書式③は、割り込みを一時停止する。以降、ファンクションキーが押されても処理ルーチンへは実行を移さない。しかし、ファンクションキーが押されたことは記憶されているので、あとでKEY ON によって割り込みを許可されると、押されたキーに対応する処理ルーチンに実行を移す。

➔「キー番号」をすべて省略すると、f・1～f・10までのすべてのファンクションキーに対して制御するとみなされる。

用　例

```
100 CONSOLE 0,25,1,1
110 ON KEY GOSUB *F1
120 KEY ON
130 PRINT "ﾌｧﾝｸｼｮﾝｷｰ 1 ｦ ｵｼﾃ ﾂｷﾞﾆ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ
ｸﾀﾞｻｲ"
140 KY$=INKEY$:IF KY$="" THEN 140
150 KEY STOP
160 PRINT "KEY STOP ｼﾏｼﾀ"
170 PRINT "ﾌｧﾝｸｼｮﾝｷｰ 1 ｦ ｵｼﾃ ﾂｷﾞﾆ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ
ｸﾀﾞｻｲ"
180 KY$=INKEY$:IF KY$="" THEN 180
190 PRINT "KEY ON ｼﾏｽ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ"
200 KY$=INKEY$:IF KY$="" THEN 200
210 KEY ON
220 PRINT "ｼｭｳﾘｮｳ ｼﾏｽ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ"
230 KY$=INKEY$:IF KY$<>" " THEN 230
240 KEY OFF
250 END
260 '-------------------------------------
270 *F1
280  PRINT
290  PRINT "*F1 ﾆ ｷﾏｼﾀ"
300  PRINT
310  RETURN
```

参　照　ON KEY GOSUB ～ RETURN

ON HELP GOSUB ～ RETURN
（オン ヘルプ ゴーサブ ～ リターン）

■HELPキーによる割り込み処理の開始行を設定

書式
① ON HELP GOSUB 処理開始行
② RETURN
③ RETURN 戻り行

➔ HELP キーを押したときに割り込みを行なう処理ルーチンの開始行を設定する。

➔この命令は割り込み処理を設定するだけで、処理の制御は HELP ON／OFF／STOP で行ない、実際の割り込み処理は HELP キーを押したときに行なわれる。

➔「処理開始行」は、行番号またはラベルで指定する。

➔割り込み処理はサブルーチンとして実行されるので、処理ルーチンからの復帰は、RETURN 文で行なう。

➔書式②は、中断したところから再開する。

➔書式③は、行番号またはラベルで指定された「戻り行」から再開する。

➔ INPUT文を実行中に割り込みが行なわれた場合、INPUT文の次の文から実行が再開される。INPUT文を正常に行なうためには、「戻り行」指定するか、割り込み処理のルーチン内で適切な処理を行なう必要があるので、注意する。

用例

```
100 CONSOLE 0,25,1,1
110 ON HELP GOSUB *HLOP
120 HELP ON
130 PRINT "HELP [redacted] [redacted] オシテ クダサイ"
140 KY$=INKEY$:[redacted] [redacted]<>" " THE[redacted]
```

```
150 HELP OFF
160 END
170 '-----------------------------------
180 *HLOP
190  PRINT
200  PRINT "HELP ｷｰ ｶﾞ ｵｻﾚﾏｼﾀ"
210  PRINT "ｼｭｳﾘｮｳ ﾊ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ"
220  PRINT
230  RETURN
```

参照　HELP　ON／OFF／STOP

HELP　ON／OFF／STOP
（ヘルプ　オン／オフ／ストップ）

HELPキーによる割り込みを制御する

書式　① HELP　ON

② HELP　OFF

③ HELP　STOP

➔ [HELP] キーが押された場合に、割り込みの許可、禁止、停止を制御する。

➔ 書式①は、割り込みを許可し、[HELP] キーが押されるたびに、ON HELP　GOSUB文で設定された処理ルーチンに実行を移す。

➔ 書式②は、割り込みを禁止し、以降、[HELP] キーが押されても処理ルーチンへは実行を移さず、通常の [HELP] キーの動作が行なわれる。

➔ 書式③は、割り込みを一時停止する。以降、[HELP] キーが押されても処理ルーチンへは実行を移さない。しかし、[HELP] キーが押されたことは記憶されているので、あとでHELP　ONによって割り込みを許可されると、指定された処理ルーチンに実行を移す。

➔プログラム終了時にHELP OFFを実行しないと、本来のヘルプ機能に戻ることができないので、注意する。

用 例

```
100 CONSOLE 0,25,1,1
110 ON HELP GOSUB *HLOP
120 HELP ON
130 PRINT "'HELP'キー ヲ オシテ ツギニ スペースバー ヲ オシテ クダサイ"
140 KY$=INKEY$:IF KY$="" THEN 140
150 HELP STOP
160 PRINT "HELP STOP シマシタ"
170 PRINT "'HELP'キー ヲ オシテ ツギニ スペースバー ヲ オシテ クダサイ"
180 KY$=INKEY$:IF KY$="" THEN 180
190 PRINT "HELP ON シマス スペースバー ヲ オシテ クダサイ"
200 KY$=INKEY$:IF KY$="" THEN 200
210 HELP ON
220 PRINT "シュウリョウ シマス スペースバー ヲ オシテ クダサイ"
230 KY$=INKEY$:IF KY$<>" " THEN 230
240 HELP OFF
250 END
260 '-----------------------------------
270 *HLOP
280  PRINT
290  PRINT "*HLOP ニ キマシタ"
300  PRINT
310  RETURN
```

参 照　ON HELP GOSUB ~ RETURN

ON　STOP　GOSUB ～ RETURN
（オン　ストップ　ゴーサブ ～ リターン）

STOPキーによる割り込み処理の開始行を設定

書式　① ON　STOP　GOSUB　処理開始行
② RETURN
③ RETURN　戻り行

➔ STOP キーを押したときに割り込み処理を行なう処理ルーチンの開始行を設定する。

➔この命令は割り込み処理を設定するだけで、処理の制御は STOP ON／OFF／STOP で行ない、実際の割り込み処理は、STOP キーを押したときに行なわれる。

➔「処理開始行」は、行番号またはラベルで指定する。

➔割り込み処理はサブルーチンとして実行されるので、処理ルーチンからの復帰は、RETURN 文で行なう。

➔書式②は、中断したところから再開する。

➔書式③は、行番号またはラベルで指定された「戻り行」から再開する。

➔ON　STOP　GOSUB文は、一般に、プログラム実行中に不用意にプログラムの実行を中断されることのないようにするために使われる。この割り込み機能がONされると、STOP キーおよび CTRL ＋C キーによるプログラムの実行中断機能が失われてしまう。そのため、プログラムにミスがなく、かつ実行中になんらかの方法でプログラムが正常に終了するようにプログラミングされているかどうかを確認してから使用するように注意しなければならない。

用 例

```
100 CONSOLE 0,25,1,1
110 ON STOP GOSUB *SPOP
120 STOP ON
130 PRINT "STOP ｷｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ"
140 PRINT "ｼｭｳﾘｮｳ ﾊ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ"
150 KY$=INKEY$:IF KY$<>" " THEN 150
160 STOP OFF
170 END
180 '-----------------------------------
190 *SPOP
200  PRINT
210  PRINT "STOP ｷｰ ｶﾞ ｵｻﾚﾏｼﾀ"
220  PRINT
230  RETURN
```

参 照　STOP ON／OFF／STOP

STOP ON／OFF／STOP
(ストップ オン／オフ／ストップ)

■STOPキー、CTRL＋Cキーによる割り込みの制御

書 式　① STOP ON
② STOP OFF
③ STOP STOP

➔ [STOP] キーが押された場合に、割り込みの許可、禁止、停止を制御する。

➔ 書式①は、割り込みを許可し、[STOP] キーが押されるたびに、ON STOP GOSUB文で設定された処理ルーチンに実行を移す。

➔ 書式②は、割り込みを禁止し、以降、[STOP] キーが押されても処理ルーチンへは実行を移さず、通常の [STOP] および [CTRL]＋Cキーの動作が行なわれる。

➔書式③は、割り込みを一時停止する。以降、STOP キーが押されても処理ルーチンへは実行を移さない。しかし、STOP キーが押されたことは記憶されているので、あとでSTOP　ONによって割り込みを許可されると、キーに対応する処理ルーチンに実行を移す。
➔プログラム終了時にSTOP　OFFを実行しないと、本来のストップ機能に戻ることができないので注意する。

用例

```
100 CONSOLE 0,25,1,1
110 ON STOP GOSUB *SPOP
120 STOP ON
130 PRINT "'STOP'ｷｰ ｦ ｵｼﾃ ﾂｷﾞﾆ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ"
140 KY$=INKEY$:IF KY$="" THEN 140
150 STOP STOP
160 PRINT "STOP STOP ｼﾏｼﾀ"
170 PRINT "'STOP'ｷｰ ｦ ｵｼﾃ ﾂｷﾞﾆ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ"
180 KY$=INKEY$:IF KY$="" THEN 180
190 PRINT "STOP ON ｼﾏｽ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ"
200 KY$=INKEY$:IF KY$="" THEN 200
210 STOP ON
220 PRINT "ｼｭｳﾘｮｳ ｼﾏｽ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ"
230 KY$=INKEY$:IF KY$<>" " THEN 230
240 STOP OFF
250 END
260 '----------------------------------
270 *SPOP
280  PRINT
290  PRINT "*SPOP ﾆ ｷﾏｼﾀ"
300  PRINT
310  RETURN
```

参照　ON　STOP　GOSUB ～ RETURN

ON TIME$ GOSUB ～ RETURN
（オン　タイムダラー　ゴーサブ ～ リターン）

■割り込み処理の発生時刻と処理の開始行を設定

書式　① ON　TIME$＝"時：分：秒"　GOSUB　処理開始行
② RETURN
③ RETURN　戻り行

➔内蔵のリアルタイムタイマによる割り込み処理の発生時刻を設定し、設定された時刻になったときに割り込み処理を行なう処理ルーチンの開始行を設定する。この命令は、割り込み処理を設定するだけで、処理の制御はTIME$　ON／OFF／STOPで行ない、実際の割り込み処理は設定された時刻になったときに行なわれる。

➔「処理開始行」は、行番号またはラベルで指定する。

➔割り込み処理はサブルーチンとして実行されるので、処理ルーチンからの復帰は、RETURN文で行なう。

➔書式②は、中断したところから再開する。

➔書式③は、行番号またはラベルで指定された「戻り行」から再開する。

➔割り込みの発生時刻は、TIME$関数の時刻設定と同様に、文字列によって、時：　分：　秒を24時間制で、それぞれ2けたの数字(午後7時8分0秒は" 19：08：00")で設定する。

➔設定された時間の分解能は2秒で、±1秒程度の誤差が生じるので注意する。

用例

```
100 CONSOLE 0,25,1,1
110 PRINT "ﾀﾀﾞｲﾏ ";TIME$;" ﾃﾞｽ ｱﾗｰﾑ ﾊ ﾅﾝｼﾞ ﾆ ｾｯﾄ ?"
120 PRINT "      **:**:** ﾉ ｶｷｶﾀ ﾃﾞ ﾄﾞｳｿﾞ"
```

```
130 INPUT "     ";TI$
140 ON TIME$=TI$ GOSUB *TMOP
150 TIME$ ON
160 PRINT "ジカン ニ ナッタラ オシラセ シマス"
170 PRINT "シュウリョウ ハ スペースバー ヲ オシテ クダサイ"
180 KY$=INKEY$
190   LOCATE 6,15:PRINT TIME$
200 IF KY$<>" " THEN 180
210 BEEP 0
220 TIME$ OFF
230 END
240 '------------------------------------
250 *TMOP
260  PRINT TI$
270  PRINT "ジカン ニ ナリマシタ スペースバー ヲ オシテクダサイ"
280  BEEP 1
290  RETURN
```

参照　TIME$　ON／OFF／STOP

TIME$　ON／OFF／STOP
(タイムダラー　オン／オフ／ストップ)

■リアルタイムタイマによる割り込みを制御する

書式　① TIME$　ON

② TIME$　OFF

③ TIME$　STOP

➔ON　TIME$　GOSUB文によって設定された割り込み発生時刻になったときに、割り込みの許可、禁止、停止を制御する。

➔書式①は、割り込みを許可し、設定された時刻になると、ON　TIME$　GOSUB文で設定された処理ルーチンに実行を移す。

➔書式②は、割り込みを禁止し、以降、割り込みは発生しない。

➔書式③は、割り込みを一時停止する。以降、設定された時刻にな

っても割り込み処理ルーチンへは実行を移さない。しかし、割り込みが発生したことは記憶されているので、あとでTIME$　ON によって割り込みを許可されると、割り込み処理ルーチンに実行を移す。

➔プログラム終了時には、TIME$　OFF を実行しておくようにする。

用例

```
100 CONSOLE 0,25,1,1
110 TI$=TIME$
120 PRINT "ﾀﾀﾞｲﾏ ";TI$;" ﾃﾞｽ"
130 SC=VAL(RIGHT$(TIME$,2))+10
140 IF 60<SC THEN PRINT "10ﾋﾞｮｳｺﾞ ﾆ ｻｲﾄﾞ RUN ｼﾃｸﾀﾞｻ
ｲ":GOTO 310
150 SC$=RIGHT$("00"+STR$(SC),2)
160 MID$(TI$,7)=SC$
170 ON TIME$=TI$ GOSUB *TMOP
180 PRINT "ｱﾗｰﾑ ｦ 10ﾋﾞｮｳｺﾞ ﾆ ｾｯﾄ ｼﾏｼﾀ"
190 TIME$ ON
200 PRINT "10 ﾋﾞｮｳ ｲｼﾞｮｳ ﾀｯﾀﾗ ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃ ｸﾀﾞｻｲ"
210 TIME$ STOP
220 PRINT "TIME$ STOP ｦ ｼﾏｼﾏﾀ"
230 PRINT "ｼﾞｶﾝ ﾆ ﾅｯﾃﾓ ｻﾌﾞﾙｰﾁﾝ ﾍﾊ ｲｷﾏｾﾝ"
240 KY$=INKEY$
250   LOCATE 6,15:PRINT TIME$
260 IF KY$<>" " THEN 240
270 TIME$ ON
280 PRINT "ｽﾍﾟｰｽﾊﾞｰ ｦ ｵｼﾃｸﾀﾞｻｲ"
290 KY$=INKEY$:IF KY$="" THEN 290
300 TIME$ OFF
310 END
320 '--------------------------------------
330 *TMOP
340  PRINT TI$;
350  PRINT " ﾆ ｱﾗｰﾑ ｾｯﾄ ｻﾚﾃ ｲﾏｼﾀ ｶﾞ"
360  PRINT TIME$;
370  PRINT " ﾆ *TMOP ﾍ ｷﾏｼﾀ"
380  RETURN
```

参照　ON　TIME$　GOSUB ～ RETURN

ERROR（エラー）

■エラー発生をシミュレートする

書式　ERROR　エラー番号

➔エラーを疑似的に発生させ、エラーのシミュレートや、エラートラップを利用したエラー処理ルーチンの呼び出しなどに利用する。

➔「エラー番号」は、0から255までの整数で指定する。

➔「エラー番号」が、BASICのエラーコードとして指定されている場合は、そのエラーをシミュレートする。つまり、システム変数ERR、ERLに「エラー番号」およびエラー発生行を設定し、プログラムの実行を中断する。

➔「エラー番号」を、BASICのエラーコードとして指定されていない未定義の番号にして、プログラム中で条件に応じたエラーを疑似的に発生させることもできる。この方法を使えば、ON ERROR GOTO文で指定されたエラー処理ルーチンへ実行を移し、エラートラップを利用した割り込み処理を行なうプログラムを組むことができる。

用例

```
100 CONSOLE 0,25,1,1
110 PRINT "ｼｭｳﾘｮｳ ﾊ 'E' ﾖ ｵｼﾃ ｸﾀﾞｻｲ"
120 PRINT
130 ON ERROR GOTO *EROP
140 INPUT "1 ｶﾗ 3 ﾉ ｶｽﾞｦ ｲﾚﾃ ｸﾀﾞｻｲ";AN$
150 IF AN$="1" THEN ERROR 101
160 IF AN$="2" THEN ERROR 102
170 IF AN$="3" THEN ERROR 103
180 IF AN$<>"E" THEN 140
190 ON ERROR GOTO 0
200 END
210 '--------------------------------------------
220 *EROP
```

```
230  IF ERR=101 AND ERL=150 THEN PRINT "ｴﾗｰｼｮﾘ
 101 ﾆ ｷﾏｼﾀ"
240  IF ERR=102 AND ERL=160 THEN PRINT "ｴﾗｰｼｮﾘ
 102 ﾆ ｷﾏｼﾀ"
250  IF ERR=103 AND ERL=170 THEN PRINT "ｴﾗｰｼｮﾘ
 103 ﾆ ｷﾏｼﾀ"
260  PRINT:PRINT
270  RESUME 140
```

参照　ERR／ERL／ON　ERROR　GOTO ～ RESUME

ERR／ERL（エラーコード／エラーライン）

■エラー番号／エラー発生行を返すシステム変数

書式　① ERR
② ERL

➔発生したエラーによって、エラーコードおよびエラーの発生した行番号が設定されるシステム変数である。

➔エラーが発生すると、ERR には発生したエラーコードが設定され、ERL にはエラーが発生した行番号が設定される。

➔一般に ERR／ERL は、ON　ERROR　GOTO 文によって指定されたエラー処理ルーチン内で、エラーの種類やエラーの発生した行の条件を判断して、適切なエラー回復処理を行なうのに利用される。

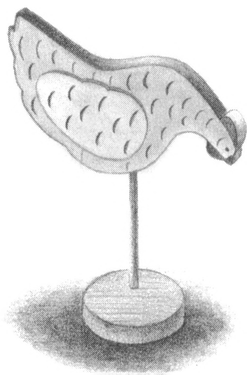
➔また、ERROR 文でエラー発生をシミュレートすることによって、積極的にエラートラップを利用したプログラムに利用することもできる。

用例

```
100 CONSOLE 0,25,1,1
110 PRINT "ｼｭｳﾘｮｳ ﾊ 'E' ｦ ｵｼﾃ ｸﾀﾞｻｲ"
120 PRINT
```

```
130 ON ERROR GOTO *EROP
140 INPUT "1 カラ 3 ノ カズヲ イレテ クダサイ";AN$
150 IF AN$="1" THEN ERROR 101
160 IF AN$="2" THEN ERROR 102
170 IF AN$="3" THEN ERROR 103
180 IF AN$<>"E" THEN 140
190 ON ERROR GOTO 0
200 END
210 '-----------------------------------
220 *EROP
230  PRINT "エラーコード ERR = ";ERR
240  PRINT "エラーギョウ ERL = ";ERL
250  IF ERR=101 AND ERL=150 THEN PRINT "エラーショリ
101 ニ キマシタ"
260  IF ERR=102 AND ERL=160 THEN PRINT "エラーショリ
102 ニ キマシタ"
270  IF ERR=103 AND ERL=170 THEN PRINT "エラーショリ
103 ニ キマシタ"
280  PRINT:PRINT
290  RESUME 140
```

参照　ON　ERROR　GOTO／ERROR



9. 文字列操作

LEFT$（レフトダラー）

■文字列左側から任意の長さの文字列を取り出す

書式　LEFT$（文字列，文字数）

➔「文字列」の左側から「文字数」で指定された長さの文字を返す関数。

➔「文字数」は、「文字列」の左端を1として数える。

➔「文字数」は0～255までの数で、「文字列」の総文字数を超えた場合は、「文字列」のすべての文字を結果として返す。

➔「文字数」が0の場合は、空（カラ）の文字列を返す。

用例

```
100 A$="ABCDEFG"
110 B$=LEFT$("123456789",5)
120 C=4
130 PRINT LEFT$(A$,2)
140 PRINT B$
150 PRINT LEFT$("ｱｲｳｴｵ",0)
160 PRINT LEFT$("123456789",C)
170 END
```

参照　RIGHT$／MID$

RIGHT$（ライトダラー）

■文字列右側から任意の長さの文字列を取り出す

書式　RIGHT$（文字列，文字数）

➔「文字列」の右側から「文字数」で指定された長さの文字を返す関数。

➔「文字数」は、「文字列」の右端を1として数える。

➔「文字数」は0～255までの数で、「文字列」の総文字数を超えた場合は、「文字列」のすべての文字を結果として返す。

➔「文字数」が0の場合は、空（カラ）の文字列を返す。

用例

```
100 A$="ABCDEFG"
110 B$=RIGHT$("123456789",3)
120 PRINT RIGHT$(A$,2)
130 C=4
140 PRINT B$
150 PRINT RIGHT$("ｱｲｳｴｵ",0)
160 PRINT RIGHT$("123456789",C)
170 END
```

参照　LEFT$／MID$



MID$（ミドルダラー）

■文字列途中から任意の長さの文字列を取り出す

書式　MID$（文字列，開始位置，文字数）

➔「文字列」の左端から数えて「開始位置」番目の文字から「文字数」で指定された長さの文字を返す関数。

➔「開始位置」は1～255の範囲で指定し、「文字列」の左端を1として数える。

➔「開始位置」が「文字列」の総文字数を超えた場合は、空（カラ）の文字列を返す。

➔「文字数」は0～255の範囲で指定し、「文字数」を省略したり、「開始位置」より右側の文字長に満たない場合は、「開始位置」より右側のすべての文字を結果として返す。

➔「文字数」が0の場合や、「文字列」の文字長より「開始位置」が大きい場合は、空（カラ）の文字列を返す。

用例

```
100 A$="ABCDEFG"
110 B$=MID$("123456789",3,5)
120 C=4:D=2
130 PRINT MID$(A$,2,3)
140 PRINT B$
150 PRINT MID$("アイウエオ",10,2)
160 PRINT MID$("アイウエオ",2)
170 PRINT MID$("アイウエオ",2,10)
180 PRINT MID$("アイウエオ",2,0)
190 PRINT MID$("アイウエオ",10,3)
200 PRINT LEFT$("123456789",0)
210 END
```

参照　RIGHT$／LEFT$

MID$（ミドルダラー）

文字列の一部を置き換える

書式 MID$（文字列，開始位置，文字数）＝置き換え文字列

➔「文字列」の「開始位置」番目の文字から、「文字数」分の文字列を、「置き換え文字列」の「文字数」分の文字列に置き換える。

➔「開始位置」は1以上、「文字列」の文字数以下の数字で指定し、「文字列」の左端を1として数えた位置を指定するので、0(空文字＝カラの文字列）を指定することはできない。

➔「文字数」が省略された場合、または「文字数」が「置き換え文字列」の文字数を超えた場合は、「置き換え文字列」のすべての文字と置き換える。

➔「文字列」の「開始位置」より右側の文字数が、「置き換え文字列」の文字数より少ない場合、「文字列」の文字数を超える部分は無視されて置き換えられないので注意する。

用例

```
100 A$="ABCDEFG"
110 B$="123456789"
120 MID$(B$,3,9)=A$
130 PRINT B$
140 B$="123456789"
150 MID$(B$,3)=A$
160 PRINT B$
170 B$="123456789"
180 MID$(B$,3,5)=A$
190 PRINT B$
200 END
```

参照 ＋（文字列の結合）／MID$（文字列の取り出し）

＋（プラス）

■文字列と文字列を結合する演算子

書式　文字列１＋文字列２

➔文字列と文字列を結合する演算子。

➔「文字列1」の後ろに「文字列2」を結合して、1つの文字列として結果を返す。

➔演算の結果は、文字変数に代入したりそのまま表示することができる。

➔演算の結果結合された文字列は、255文字以下でなければならない。

用例

```
100 A$="ABCDEFG"
110 B$="123456789"
120 D$=A$+B$
130 PRINT D$
140 PRINT A$+B$
150 PRINT A$;B$
160 END
```

参照　MID$

ASC（アスキー）

■文字のキャラクタコードを返す

書式　ASC（文字列）

➔「文字列」の最初の1文字のキャラクタコードを返す。

用例

```
100 A$="ABCDEFG"
110 B$="123456789"
120 C$=(A$)
130 PRINT C$
140 PRINT ASC(A$)
150 PRINT ASC(B$)
160 PRINT ASC("A")
170 END
```

参照　CHR$／22ページ「アスキーコード表」

CHR$（キャラクタダラー）

■キャラクタコードに対応する文字を返す

CHR$（キャラクタコード）

➔指定された「キャラクタコード」に対応する文字を返す。

➔「キャラクタコード」は、0～255の範囲の数値で指定する。

用例

```
100 A=49
110 PRIN[redacted] CH[redacted]$(80)
120 PRIN[redacted] CH[redacted]$(A)
130 END
```

参照　ASC／22ページ「アスキーコード表」

STR$（エスティーアールダラー）

■数値データを文字型データに変換する

書式　STR$（数値）

➔数値データを文字型のデータに変換する。

➔「数値」は変数や式の形で与えることができる。この場合、変数の内容や式の計算結果が文字型データに変換される。

➔STRING$（ストリングダラー）と混同しないように注意する。

用例

```
100 A=49
110 PRINT STR$(80)
120 PRINT STR$(A)
130 END
```

参照　VAL／STRING$

VAL（バリュー）

■数字の文字列データを数値データに変換する

書式　VAL（文字列）

➔数字の文字列で構成されているデータを、数値型のデータに変換する。

➔「文字列」は、最初の1文字が「+、−、&、または数字」でなければ、変換した結果は0（ゼロ）が返される。

➔文字列に含まれるスペースは無視されるので、注意する。

用例

```
100 A$="123"
110 PRINT VAL("-123")
120 PRINT VAL(A$)
130 PRINT VAL("ABC")
140 END
```

参照　STR$

LEN（レングス）

■文字列の文字数を返す

書式　LEN（文字列）

➔「文字列」で指定された文字型データの文字数を、数値型のデータで返す。

➔「文字列」にはディスプレイに表示されないコントロールコード（キャラクタコードで0～31までのコードのことをいい、ディスプレイや他の機器を制御する）やスペースも1文字として数える。

用例

```
100 A$="123456"
110 PRINT LEN("ABCD")
120 PRINT LEN(A$)
130 PRINT LEN("A   E")
140 END
```

参照　KLEN

INSTR（インストリング）

■指定した文字列を探してその位置を返す

書式　① INSTR（被検索文字列，検索文字列）
　　　　② INSTR（開始位置，被検索文字列，検索文字列）

➔「被検索文字列」の中から「検索文字列」で指定した文字列を探し、見つかった場合は、その位置を左端からの文字数で返し、見つからなかった場合は0をそれぞれ数値で返す。

➔書式①は、「被検索文字列」の初めから探し始める。

➔書式②は、「被検索文字列」の中の探し始める位置である「開始位置」を左端からの文字数で指定する。

➔検索文字列に空（カラ）の文字列を指定すると、「開始位置」と同じ値を返す。

➔「開始位置」は、変数や式の結果として与えることもできる。

用例

```
100 A$="123ABC456"
110 PRINT INSTR("ABCDEFGHI","DEF")
120 PRINT INSTR(5,"ABCDEFGHI","DEF")
130 PRINT INSTR(A$,"ABC")
140 PRINT INSTR(A$,"")
150 END
```

参照　KINSTR

SPACE$（スペースダラー）

■指定した長さの空白文字列を返す

書式　SPACE$（文字数）

➔「文字数」で指定した数だけの、空白（スペース）を並べた文字列を返す。

➔「文字数」は、0～255の範囲で指定しなければならない。

➔ SPC と混同しないように注意する。

➔「文字数」は、変数や式の結果として与えることもできる。

用例

```
100 A$=SPACE$(5)
110 PRINT A$
120 PRINT "<";A$;">"
130 PRINT "<";SPACE$(5);">"
140 PRINT LEN(A$)
150 END
```

参照　SPC／STRING$

SPC（スペース）

■指定された数だけの空白を出力する

書式　SPC（文字数）

➔ PRINT や LPRINT 文などの出力文中に限り使用できる関数で、「文字数」で指定された数の空白（スペース）を出力する。

➔「文字数」は、0～現在の水平表示文字数、または水平印字文字数で指定する。

➔「文字数」として指定できる範囲は－32768から32767までの数値

が許されるが、負の数を指定するとすべて0（ゼロ）とみなす。

➔また、現在の水平表示文字数、または水平印字文字数を超えて指定した場合は、「文字数」を現在の水平表示文字数、または水平印字文字数で割った余りを、値として指定したとみなされる。

➔SPACE$と混同しないように注意する。

➔「文字数」は、変数や式の結果として与えることもできる。

用例

```
100 PRINT SPC(5);"ABCD"
110 PRINT SPC(5);"<";">"
120 PRINT "<";SPC([illegible]);">"
130 PRINT "<";SPC(180);">"
140 END
```

参照　SPACE$／TAB

TAB（タビュレーション）

■指定した位置まで空白を出力しカーソルを移動

書式　TAB（文字数）

➔PRINTやLPRINT文などの出力文中で、現在カーソルのある行の左端から「文字数」で指定した位置まで空白を出力し、カーソルを移動する。

➔「文字数」は、－32768～32767までの数値で指定することが許されるが、負の指定はすべて0（ゼロ）とみなされ、左端を指定したことになる。

➔また、現在の水平表示文字数、または水平印字文字数を超えて指定した場合は、「文字数」を現在の水平表示文字数、または水平印字文字数で割った余りを、値として指定したとみなされる。

➔SPCと混同しないように注意する。

➔「文字数」は、変数や式の結果として与えることもできる。

用例

```
100 PRINT TAB(5);"ABCD"
110 PRINT TAB(5);"<";TAB(40);">"
120 PRINT TAB(5);"ABC";TAB(-10);"DEF"
130 PRINT TAB(180);"ABC"
140 END
```

参照　SPACE$／SPC

STRING$（ストリングダラー）

■指定した文字を指定した数だけ返す

書式　①STRING$（文字数，文字）
②STRING$（文字数，キャラクタコード）

➔書式①は、「文字」で指定した文字を、「文字数」で指定した数だけ並べた文字列を返す。
➔書式②は、0～255の「キャラクタコード」で指定した文字を、「文字数」で指定した数だけ並べた文字列を返す。
➔指定した「文字」が、複数の文字による文字列であった場合、文字列の最初の1文字が指定されたとみなされる。
➔「文字数」は、変数や式の結果として与えることもできる。

用例

```
100 A$="-"
110 PRINT STRING$(20,"*")
120 PRINT STRING$(20,A$)
130 PRINT "<";STRING$(5," ");">"
140 END
```

参照　STR$／SPACE$／SPC／TAB

HEX$（ヘキサデシマルダラー）

10進数の数値データを16進数の文字列に変換

書式　HEX$（数値データ）

➔「数値データ」を16進数に変換して、文字型のデータとして返す。

➔「数値データ」は、−32768〜32767の範囲で与えなければならない。結果は０〜FFFFに変換される。

➔また、32768〜65535の範囲で指定することもできるが、結果は、8000〜FFFFに変換される。

➔「数値データ」は、変数や式の結果として与えることもできる。

用例

```
100 A=1234
110 PRINT HEX$(0)
120 PRINT HEX$(16)
130 PRINT HEX$(A)
140 PRINT HEX$(32767)
150 PRINT HEX$(65535!)
160 PRINT HEX$(-32768!)
170 PRINT HEX$(456-123)
180 END
```

参照　VAL／OCT$

OCT$（オクタルダラー）

10進数の数値データを8進数の文字列に変換

書式　OCT$（数値データ）

➔「数値データ」を8進数に変換して、文字型のデータとして返す。

➔「数値データ」は、−32768～32767の範囲で与えなければならない。結果は 0 ～177777に変換される。

➔また、32768～65535の範囲で指定することもできるが、結果は、100000～177777に変換される。

➔「数値データ」は、変数や式の結果として与えることもできる。

用例

```
100 A=1234
110 PRINT OCT$(0)
120 PRINT OCT$(8)
130 PRINT OCT$(A)
140 PRINT OCT$(32767)
150 PRINT OCT$(65535!)
160 PRINT OCT$(-32768!)
170 PRINT OCT$(456-123)
180 END
```

参照　VAL／HEX$

AKCNV$（アスキー漢字コンバートダラー）

アスキー文字を対応する日本語文字に変換

書式　AKCNV$（文字列）

➔「文字列」で指定された文字列のうちから、1バイト系の文字を2バイト系の日本語文字列に変換する。

➔変換が行なわれるのは、数字、アルファベット、カナなどで、キャラクタコードと日本語文字コードが対応しているものに限られる。

用例

```
100 A$="ABC"
110 PRINT A$
120 PRINT AKCNV$(A$)
130 PRINT AKCNV$("ABCDEＡＢＣＤＥ")
140 PRINT AKCNV$("ｱｲｳｴｵ")
150 PRINT AKCNV$("+-*/¥=")
160 END
```

参照　KACNV$

KACNV$（漢字アスキーコンバートダラー）

漢字を対応するアスキー文字に変換

書式　KACNV$（日本語文字列）

➔「日本語文字列」で指定した文字列のうちから、2バイト系の日本語文字列を1バイト系のアスキー文字列に変換する。

➔変換が行なわれるのは、数字、アルファベット、カナなどで、キャラクタコードと日本語文字コードが対応しているものに限られる。

用例

```
100 A$="ＡＢＣ"
110 PRINT A$
120 PRINT KACNV$(A$)
130 PRINT KACNV$("ＡＢＣＤABCDE")
140 PRINT KACNV$("アイウエオ")
150 PRINT KACNV$("＋－＊／￥＝")
160 END
```

参照　AKCNV$

JIS$（ジスダラー）

■日本語文字列を JIS 漢字コードに変換

書式　JIS$（日本語文字列）

➔「日本語文字列」の最初の2バイトを、JIS 漢字コードの16進表記の4桁の文字列として返す。

➔日本語文字列の始まりと終わりには、必ず「KI（&h1B4B）と「KO（&h1B48）」コードが付加されているので、一般的には漢字の2文字目を指定して変換する。

用例

```
100 A$="漢字"
110 PRINT A$
120 PRINT JIS$(A$)
130 PRINT JIS$(KMID$(A$,2,1))
140 PRINT KLEN(A$)
150 END
```

参照　KNJ$／KMID$

KNJ$（漢字ダラー）

■JIS 漢字コードを日本語文字列に変換

書式　KNJ$（漢字コード文字列）

➔「漢字コード文字列」の最初の16進表記、4桁の文字列を、JIS 漢字コードとして日本語文字列に変換して返す。

➔「漢字コード文字列」は、4桁の0～9、A～F の16進表記の文字を使い、漢字コードまたは「KI／KO コード」で指定する。

用　例

```
100 A$="3441"
110 KI$="1B4B"
120 KO$="1B48"
130 PRINT A$
140 PRINT KNJ$("1B4B")+KNJ$(A$)+KNJ$("1B48")
150 PRINT KNJ$(KI$)+KNJ$("3B7A")+KNJ$(KO$)
160 PRINT LEN(A$)
170 END
```

参　照　　JIS$

KLEN（漢字レングス）

■日本語を含む文字列の文字数を返す

書　式　　KLEN（文字列，機能番号）

➔日本語を含む「文字列」のうち「機能番号」で指定されたタイプの文字数を返す。機能番号は0～5の数値で指定する。

0………全体文字数(日本語文字は2バイト1文字、KI／KO含む)
1………1バイト系アスキー文字（英数カナ文字）の文字数
2………2バイト系日本語の文字数(KI／KOコードを含まない)
3………2バイト系全角文字だけの文字数
4………2バイト系半角文字だけの文字数
5………KI／KOコードの数

➔「機能番号」を省略すると0（ゼロ）を指定したとみなされる。

用　例

```
100 A$="漢字"
110 PRINT A$
120 PRINT KLEN(A$)
130 PRINT LEN(A$)
140 END
```

参　照　　LEN

KEXT$（漢字イクストラクトダラー）

アスキー文字、日本語文字を抜き出して返す

書式　①　KEXT$（文字列，0）
　　　　②　KEXT$（文字列，1）

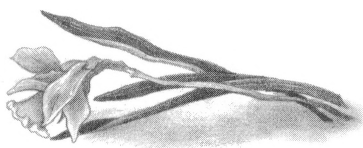
➔書式①は、「文字列」の中から1バイト系のアスキー文字（英数カナ文字）だけを抜き出して返す。

➔書式②は、「文字列」の中から2バイト系の日本語文字列だけを抜き出して返す。このとき「KI／KO コード」は含まれないので注意する。

➔抜き出すべき文字がないと、空（カラ）の文字列を返す。

用例

```
100 A$="ABCD漢字EFGH"
110 PRINT A$
120 PRINT KEXT$(A$,0)
130 PRINT KEXT$(A$,1)
140 PRINT KEXT$("123日本語456",0)
150 PRINT KEXT$("123日本語456",1)
160 END
```



KINSTR（漢字インストリング）

■日本語文字列中の指定文字列の位置を返す

書式　① KINSTR（被検索文字列，検索文字列）
　　　　② KINSTR（開始位置，被検索文字列，検索文字列）

➔日本語を含む文字列の中から、指定した文字列を探して、その文字列の位置を返す。

➔「被検索文字列」の中から「検索文字列」で指定した文字列を探し、見つかった場合は、その位置を初めからの文字数で返し、見つからなかった場合は0をそれぞれ数値で返す。

➔書式①は、「被検索文字列」の初めから探し始める。

➔書式②は、「被検索文字列」の中の探し始める位置である「開始位置」を左端からの文字数で指定する。

➔「開始位置」は、日本語の文字も1文字と数え、「KI／KOコード」も1文字と数えて指定する。

➔検索文字列に空（カラ）の文字列を指定すると、「開始位置」と同じ値を返す。

➔「開始位置」は、変数や式の結果として与えることもできる。

用例

```
100 A$="ABCD漢字EFGH"
110 PRINT A$
120 PRINT KINSTR(A$,"字")
130 PRINT KINSTR("ABC日本語の漢字DEF","語の")
140 PRINT KINSTR(7,"ABC日本語の漢字DEF","語の")
150 END
```

参照　LEN

KTYPE（漢字タイプ）

■日本語文字列中の指定位置の文字のタイプを返す

書式　KTYPE（文字列，位置）

➔日本語を含む「文字列」の中から、指定した「位置」番目の文字のタイプを返す。

➔「KI／KO コード」も1文字と数える。

➔タイプは0～4の数値で返され、それぞれ次の意味を持つ。

0………アスキー文字（1バイト系英数カナ文字）
1………全角日本語文字（2バイト系全角）
2………半角日本語文字（2バイト系半角）
3………KI コード（漢字 IN）
4………KO コード（漢字 OUT）

➔「文字列」に空（カラ）の文字列や、「位置」に0、または「文字列」の文字数より大きな数は指定できないので、注意する。

用例

```
100 A$="ABCD漢字EFGH"
110 PRINT A$
120 PRINT KTYPE(A$,6)
130 PRINT KTYPE("ABC日本語の漢字DEF",1)
140 PRINT KTYPE("ABC日本語の漢字DEF",6)
150 PRINT KTYPE("ABC日本語の漢字DEF",12)
160 PRINT KTYPE("ABC日本語の漢字DEF",4)
170 PRINT KTYPE("ABC日本語の漢字DEF",11)
180 END
```

KMID$（漢字ミドルダラー）

■日本語文字列中の任意の文字列を取り出す

書　式　　KMID$（文字列，開始位置，文字数）

➔日本語を含む文字列の途中から、任意の長さの文字列を取り出す関数。「文字列」の左端から数えて「開始位置」番目の文字から、「文字数」で指定された長さの文字を返す。「開始位置」は1～255の範囲で指定し、「文字列」の左端を1として数える。

➔「開始位置」が、「文字列」の総文字数を超えた場合は、空（カラ）の文字列を返す。

➔「文字数」は、0～255の範囲で指定し、「文字数」を省略したり、「開始位置」より右側の文字長に満たない場合は、「開始位置」より右側のすべての文字を結果として返す。

➔KMID$関数で数える文字数は、アスキー文字（1バイト系英数カナ文字）は1バイト1文字、日本語文字（2バイト系）および「KI／KOコード」は2バイトを1文字として数える。

➔「文字数」が0の場合や、「文字列」の文字長より「開始位置」が大きい場合は、空（カラ）の文字列を返す。

用　例

```
100 A$="ABCD漢字EFGH"
110 PRINT A$
120 PRINT KMID$(A$,6,2)
130 PRINT KMID$(A$,5,4)
140 PRINT KMID$(A$,2,2)
150 PRINT KMID$("ABC日本語の漢字DEF",4,3)
160 PRINT KMID$("ABC日本語の漢字DEF",2,2)
170 END
```

参　照　　MID$

10. 数値関数

ABS（アブソリュート）

■絶対値を返す

書式　ABS（数値データ）

➔与えられた「数値データ」の絶対値を求める。

➔結果は、値の符号が正の場合はそのまま、負の場合は符号が除かれた値となる。

➔結果は単精度実数で返されるが、「数値データ」が倍精度実数の場合は、倍精度実数の値で返される。

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用例

```
100 A=123
110 B=-456
120 PRINT A
130 PRINT B
140 PRINT ABS(A)
150 PRINT ABS(B)
160 PRINT ABS(222-111)
170 PRINT ABS(111-222)
180 C=ABS(A+B)
190 PRINT C
200 END
```

参照　SGN

SGN（サイン）

■数値データの符号を返す

書式　SGN（数値データ）

➔与えられた「数値データ」の符号を調べる。

➔結果は、次の値で返される。

　正の場合（数値データ>0）………1

　0の場合（数値データ=0）………0

　負の場合（数値データ<0）………−1

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用例

```
100 A=123
110 B=0
120 C=-456
130 PRINT A
140 PRINT B
150 PRINT C
160 PRINT SGN(A)
170 PRINT SGN(B)
180 PRINT SGN(C)
190 PRINT SGN(222-111)
200 D=SGN(A+C)
210 PRINT D
220 END
```

参照　ABS

INT（インテジャー）

■小数点以下を切り捨てた整数値を返す

書式　INT（数値データ）

➔与えられた「数値データ」の小数点以下を切り捨てた値、すなわち、「数値データ」を超えない最大の整数を返す。

➔ INTは、「数値データ」の値が負の場合に、「数値データ」より小さい最大の整数を返すのに対し、FIXの場合は、単に「数値データ」の小数点以下の部分を取り除いただけの値を返す。この違いに注意する。

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用例

```
100 A=123.456
110 B=234.567
120 C=-123.456
130 PRINT A
140 PRINT B
150 PRINT C
160 PRINT INT(A)
170 PRINT INT(B)
180 PRINT INT(C)
190 PRINT INT(A+.5)
200 PRINT INT(B+.5)
210 PRINT INT(C)
220 PRINT FIX(C)
230 END
```

参照　FIX

FI[redacted] ([redacted]ックス)

■整数部分を返す

書式　FIX（数値データ）

➔ 与えられた「数値データ」の小数点以下を取り除いて、整数部分だけを取り出した値を返す。

➔ FIX は、「数値データ」の値が負の場合に、単に小数点以下の部分を取り除いただけの値を返すが、INT の場合は、「数値データ」より小さい最大の整数を返す。この違いに注意する。

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用例

```
100 A=123.456
110 B=234.567
120 C=-123.456
130 PRINT A
140 PRINT B
150 PRINT C
160 PRINT FIX(A)
170 PRINT FIX(B)
180 PRINT FIX(C)
190 PRINT FIX(C)
200 PRINT INT(C)
210 END
```

参照　INT

CINT／CSNG／CDBL
（コンバート　インテジャー／シングル／ダブル）

■整数値、単精度実数値、倍精度実数値に変換する

書式
① CINT（数値データ）
② CSNG（数値データ）
③ CDBL（数値データ）

➔書式①は、単精度実数値と倍精度実数値を、整数値に変換する。変換は、「数値データ」の小数点以下を四捨五入して整数にする。結果は、−32768〜32767の範囲になければならない。

➔書式②は、整数値と倍精度実数値を、単精度実数値に変換する。変換は、有効桁6桁で行なわれる。結果の値は、−1.70141 E＋38〜1.70141 E＋38の範囲になければならない。

➔書式③は、整数値と単精度実数値を倍精度実数値に変換する。変換は行なわれるが、有効桁は変化しないので、得られた値の精度は変換前の値と同じになる。

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用例

```
100 A%=123
110 B!=456.789
120 C#=7890123.456#
130 PRINT A%
140 PRINT B!
150 PRINT C#
160 PRINT CINT(B!)
170 PRINT CSNG(C#)
180 PRINT CDBL(A%)
190 END
```

SIN（サイン）

■サイン（正弦）を返す

書式　SIN（数値データ）

➔与えられた「数値データ」をラジアン角として、正弦（sin　X）値を返す。

➔結果は単精度実数で返されるが、「数値データ」が倍精度実数の場合は、倍精度実数の値で返される。

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用例

```
100 SCREEN 1,0,0,7:CLS 3
110 A=60
120 B#=30.123456789#
130 PAI=3.14159
140 PRINT SIN(A)
150 PRINT SIN(B#)
160 VIEW (200,50)-(639,149)
170 WINDOW (0,-1)-(720,1)
180 LINE (0,0)-(720,0)
190 FOR X=0 TO 720 STEP 10
200   Y=SIN(X*2/PAI)
210   PSET (X,Y),7
220 NEXT X
230 END
```

参照　COS／TAN／ATN

COS（コサイン）

■コサイン（余弦）を返す

書式　COS（数値データ）

➔与えられた「数値データ」をラジアン角として、余弦（cos X）値を返す。

➔結果は単精度実数で返されるが、「数値データ」が倍精度実数の場合は、倍精度実数の値で返される。

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用例

```
100 SCREEN 1,0,0,7:CLS 3
110 A=60
120 B#=30.123456789#
130 PAI=3.14159
140 PRINT COS(A)
150 PRINT COS(B#)
160 VIEW (200,50)-(639,149)
170 WINDOW (0,-1)-(720,1)
180 LINE (0,0)-(720,0)
190 FOR X=0 TO 720 STEP 10
200   Y=COS(X*2/PAI)
210   PSET (X,Y),7
220 NEXT X
230 END
```

参照　SIN／TAN／ATN

TAN（タンジェント）

タンジェント（正接）を返す

書式　TAN（数値データ）

➔与えられた「数値データ」をラジアン角として、正接（tan　X）値を返す。

➔結果は単精度実数で返されるが、「数値データ」が倍精度実数の場合は、倍精度実数の値で返される。

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用例

```
100 SCREEN 1,0,0,7:CLS 3
110 A=60
120 B#=30.123456789#
130 PAI=3.14159
140 PRINT TAN(A)
150 PRINT TAN(B#)
160 VIEW (200,50)-(599,149)
170 WINDOW (0,-10)-(360,10)
180 LINE (0,0)-(360,0)
190 FOR X=0 TO 360 STEP 10
200   IF (X+90) MOD 180=0 THEN 230
210   Y=TAN(X*2/PAI)
220   PSET (X,Y),7
230 NEXT X
240 END
```

参照　SIN／COS／ATN

ATN（アークタンジェント）

■アークタンジェント（逆正接）を返す

書式　ATN（数値データ）

➔与えられた「数値データ」をラジアン角として、逆正接（arctan X）値を返す。

➔結果として得られる値は、$-\pi/2\sim\pi/2$の範囲になる。

➔結果は単精度実数で返されるが、「数値データ」が倍精度実数の場合は、倍精度実数の値で返される。

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用例

```
100 A=60
110 B#=30.123456789#
120 PRINT ATN(A)
130 PRINT ATN(B#)
140 END
```

参照　SIN／COS／TAN

SQR（スクェア　ルート）

■ルート（平方根）を返す

書式　SQR（数値データ）

➔与えられた「数値データ」の平方根（$\sqrt{X}$）を求めて返す。

➔結果は単精度実数で返されるが、「数値データ」が倍精度実数の場合は、倍精度実数の値で返される。

➔「数値データ」には、変数の内容や式の計算結果を与えることが

できる。

用 例

```
100 A=100
110 B#=1230.123456789#
120 C=2
130 D#=2#
140 PRINT SQR(A)
150 PRINT SQR(B#)
160 PRINT SQR(C)
170 PRINT SQR(D#)
180 END
```

EXP（イクスポーネンシャル）

■指数関数の値を返す

書 式　　EXP（数値データ）

➔与えられた「数値データ」のeに対する指数関数（eの「数値データ」乗）を求めて返す。

➔結果は単精度実数で返されるが、「数値データ」が倍精度実数の場合は、倍精度実数の値で返される。

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用 例

```
100 A=5
110 B#=12.123456789#
120 C=2
130 D#=2#
140 PRINT EXP(A)
150 PRINT EXP(B#)
160 PRINT EXP(C)
170 PRINT EXP(D#)
180 END
```

LOG（ログ）

■自然対数（ログ）を返す

書式　LOG（数値データ）

➔与えられた「数値データ」の自然対数（log eX）を求めて返す。

➔結果は単精度実数で返されるが、「数値データ」が倍精度実数の場合は、倍精度実数の値で返される。

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用例

```
100 A=5000
110 B#=12345678.12345#
120 C=2
130 D#=2#
140 PRINT LOG(A)
150 PRINT LOG(B#)
160 PRINT LOG(C)
170 PRINT LOG(D#)
180 END
```

RND（ランダム）

■乱数を返す

書式　RND（数値データ）

➔0以上1未満の乱数を発生して返す。

➔発生する乱数系列は、RUNコマンド、CLEAR文が実行されるたびに同じ系列をとる。

➔発生する乱数系列は、RANDOMIZE文によって変更できる。

➔「数値データ」によって、発生する乱数を指定することができる。

「数値データ」＝負………乱数系列を初期化する。

「数値データ」＝０………ひとつ前に発生した乱数の値をとる。

「数値データ」＝正………次の新しい乱数を発生する。

➔「数値データ」が省略されると、正の値を指定されたとみなされる。

➔「数値データ」には、変数の内容や式の計算結果を与えることができる。

用　例

```
100 FOR I=1 TO 12
110 LOCATE 0,I
120   PRINT RND(-1)
130 NEXT I
140 PRINT:PRINT:PRINT
150 FOR I=1 TO 12
160 LOCATE 15,I
170   PRINT RND(0)
180 NEXT I
190 PRINT:PRINT:PRINT
200 FOR I=1 TO 12
210 LOCATE 30,I
220   PRINT RND(1)
230 NEXT I
240 PRINT:PRINT:PRINT
250 FOR I=1 TO 12
260 LOCATE 45,I
270   PRINT INT(RND(1)*10)+1,
280 NEXT I
290 END
```

参　照　　RANDOMIZE／RUN／CLEAR

●N88-日本語BASIC(86)エラーコード表

コード	エラーメッセージ	意　味
1	NEXT without FOR	FORに対応しないNEXTがある
2	Syntax error	文の記述が文法にあっていない
4	Out of DATA	読むべきデータがないのにREAD文が実行された
5	Illegal function call	関数や命令の機能の指定のしかたがまちがっている
8	Undefined line number	GOTO文やGOSUB文の飛び先の行番号が存在しない
9	Subscript out of range	配列変数の添え字が指定した範囲を超えている
10	Duplicate Definition	配列またはユーザー定義関数を二重に定義した
13	Type mismatch	関数の引数、式の左右辺などの型が一致しない
17	Can't continue	CONTコマンドで実行を再開できない
18	Undefined user function	DEF FN文で定義されていないユーザー定義関数を使った
22	Missing operand	命令の中に必要なパラメータが指定されていない
26	FOR whithout NEXT	NEXTに対応しないFORがある
31	Duplicate label	同じラベル名が2つ以上ある
32	Undefined label	定義されていないラベル名を使った
55	Input past end	ファイル中のすべてのデータを読み終わったのに、さらに読み出そうとした
56	Bad file name	ファイル名の指定がまちがっている

※このエラーコード表は、エラーの中からよく出会う一部のものを、ERR関数で返されるエラー番号の順に並べたものである。

索　　引（ABC順）

< I >

< J >

< K >

< L >

目的別索引（あいうえお順）

PC-98シリーズ
BASIC用語・用例辞典

著　者　　伊　東　ひ　ろ　み
発行者　　深　見　兵　吉
印刷所　　広研印刷株式会社

発　行　所
成美堂出版
〒112　東京都文京区水道1－8－2
電話(03)3814－4351 振替・東京7-4466

ISBN4-415-07726-9

●成美堂出版の本

MAC活用事典　UNIZON編著
DOS/V活用事典　UNIZON編著
ワープロ パソコン 通信データブック　成美堂出版編
すぐわかるワープロ・パソコン用語辞典　大藪多可志監修
すぐ使えるトラベル英会話　ブラック千尋監修
すぐに役立つ中国語会話　田中亮三郎著
すぐに役立つ韓国語会話　李 廣善著
すぐに役立つタイ語会話　K・ドゥアンヌットほか著
見せれば通じる七ヵ国語会話　海外旅行研究会編
まんが初めての海外旅行　戸山良男監修
まんが海外旅行で役立つ英会話　金子良三監修
まんが女性のための英会話　ブラック千尋監修
まんがヨーロッパ旅行の英会話　金子良三監修

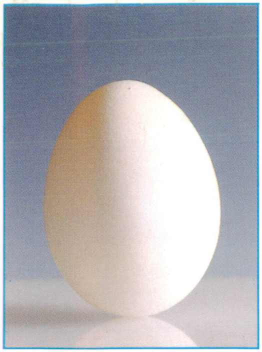
●趣味、教養●

まんがアメリカ旅行の英会話　金子良三監修
まんが初めてのホームステイ　金子良三監修
やさしい英文手紙の書き方　金子良三監修
英文 短い手紙の文例集　藤井亮蔵ほか著
CD付 中国語会話事典　田中亮三郎著
CD付 韓国語会話事典　李 廣善著
CD付 海外旅行の英会話事典　水野修身監修
毎日活用 日用語字典　大河内昭爾監修
平成版 カタカナ語新辞典　成美堂出版編集部編
ことわざ活用辞典　成美堂出版編集部編
ポケット版 外来語新語辞典　矢ケ崎誠治監修
ポケット版 常用語字典　大河内昭爾監修
ポケット版 ことわざ辞典　槌田満文監修

PC-98シリーズ

BASIC用語・用例辞典

発　行	1995年4月20日
定　価	680円(本体660円)
著　者	伊東ひろみ
発行者	深見兵吉
印刷所	広研印刷株式会社
発行所	成美堂出版 〒112 東京都文京区水道1-8-2 ☎(03)3814-4351 FAX3814-4355



PC-98シリーズ

BASIC 用語・用例辞典

9784415077260

1912355006806

ISBN4-415-07726-9

C2355 P680E

定価680円(本体660円)